System Interfaces Migration Information

This document contains a section for each system interface defined in XSH, Issue 5. Each section lists the synopsis and identifies syntax and semantic changes made to the interface in Issue 5 (if any), complete with examples where appropriate. Only changes that might affect an application programmer are identified.

a64l( ), l64a( )

Convert between a 32-bit integer and a radix-64 ASCII string.

```
#include <stdlib.h>

long a64l(const char *s);
char *l64a(long value);
```

Issue 5: A note indicating that these interfaces need not be reentrant is added to the Description. A portable multi-threaded application may only safely call these functions when access to the function is serialized. For example:

```
...
/*
 * An example of thread-safety by serializing access
 * to a non-reentrant function.
 */
pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;

void my_l64a( long value, char *buffer)
{
int i=0;
char *ptr;

(void) pthread_mutex_lock (&mtx);
ptr = l64a( value);
(void) strcpy (buffer, ptr);
(void) pthread_mutex_unlock (&mtx);
}
...
```

abort()

Generate an abnormal process abort.

```
#include <stdlib.h>

void abort(void);
```

Issue 5: No functional changes in this issue.

abs( )

Return an integer absolute value.

```
#include <stdlib.h>

int abs(int i);
```

Issue 5: No functional changes in this issue.

access( )

Determine accessibility of a file.

```
#include <unistd.h>

int access(const char *path, int amode);
```

Issue 5: No functional changes in this issue.

acos( )

Arc cosine function.

```
#include <math.h>

double acos(double x);
```

Issue 5: No functional changes in this issue.

acosh( ), asinh( ), atanh( )

Inverse hyperbolic functions.

```
#include <math.h>

double acosh(double x);
double asinh(double x);
double atanh(double x);
```

Issue 5: No functional changes in this issue.

advance( )

Pattern match given a compiled regular expression. (**LEGACY**)

```
#include <regexp.h>

int advance(const char *string, const char *expbuf);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations. New applications should use the *regcomp*( ), *regexec*( ), *regerror*( ), and *regfree*( ) functions, and the **<regex.h>** header, which provide full internationalized regular expression functionality compatible with POSIX.2 and XBD, Chapter 7, *Regular Expressions*.

aio_cancel( )

Cancel an asynchronous I/O request. (**REALTIME**)

```
#include <aio.h>

int aio_cancel(int fildes, struct aiocb *aiocbp);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *aio_cancel*( ) function attempts to cancel one or more asynchronous I/O requests currently outstanding against file descriptor *fildes*. The *aiocbp* argument points to the asynchronous I/O control block for a particular request to be canceled.

aio_error( )

Retrieve errors status for an asynchronous I/O operation. (**REALTIME**)

```
#include <aio.h>

int aio_error(const struct aiocb *aiocbp);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *aio_error*( ) function returns the error status associated with the **aiocb** structure referenced by the *aiocbp* argument. The error status for an asynchronous I/O operation is the *errno* value that would be set by the corresponding *read*( ), *write*( ), or *fsync*( ) operation.

aio_fsync( )

Asynchronous file synchronization. (**REALTIME**)

```
#include <aio.h>

int aio_fsync(int op, struct aiocb *aiocbp);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *aio_fsync*( ) function asynchronously forces all I/O operations associated with the file indicated by the file descriptor *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument and queued at the time of the call to *aio_fsync*( ) to the synchronized I/O completion state. The function call returns when the synchronization request has been initiated or queued to the file or device (even when the data cannot be synchronized immediately).

aio_read( )

Asynchronous read from a file. (**REALTIME**)

```
#include <aio.h>

int aio_read(struct aiocb *aiocbp);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *aio_read*( ) function allows the calling process to read *aiocbp→aio_nbytes* from the file associated with *aiocbp→aio_fildes* into the buffer pointed to by *aiocbp→ aio_buf*. The function call returns when the read request has been initiated or queued to the file or device (even when the data cannot be delivered immediately).

Large File Support extensions are added; see Chapter 17 on page 163.

aio_return( )

Retrieve return status of an asynchronous I/O operation. (**REALTIME**)

```
#include <aio.h>

ssize_t aio_return(struct aiocb *aiocbp);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *aio_return*( ) function returns the return status associated with the **aiocb** structure referenced by the *aiocbp* argument. The return status for an asynchronous I/O operation is the value that would be returned by the corresponding *read*( ), *write*( ), or *fsync*( ) function call.

aio_suspend( )

Wait for an asynchronous I/O request. (**REALTIME**)

```
#include <aio.h>

int aio_suspend(const struct aiocb * const list[], int nent,
    const struct timespec *timeout);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *aio_suspend*( ) function suspends the calling thread until at least one of the asynchronous I/O operations referenced by the *list* argument has completed, until a signal interrupts the function, or, if *timeout* is not NULL, until the time interval specified by *timeout* has passed.

aio_write( )

Asynchronous write to a file. (**REALTIME**)

```
#include <aio.h>

int aio_write(struct aiocb *aiocbp);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *aio_write*( ) function allows the calling process to write *aiocbp→aio_nbytes* to the file associated with *aiocbp→aio_fildes* from the buffer pointed to by *aiocbp→aio_buf*. The function call returns when the write request has been initiated or, at a minimum, queued to the file or device.

Large File Support extensions are added; see Chapter 17 on page 163.

alarm( )

Schedule an alarm signal.

```
#include <unistd.h>

unsigned int alarm(unsigned int seconds);
```

Issue 5: No functional changes in this issue.

asctime, asctime_r( )

Convert date and time to a string.

```
#include <time.h>

char *asctime(const struct tm *timeptr);
char *asctime_r(const struct tm *tm, char *buf);
```

Issue 5: No functional changes in this issue.

The *asctime_r*( ) function is a new function included for alignment with the POSIX Threads Extension. It is the thread-safe equivalent of the *asctime*( ) function.

A note indicating that the *asctime*( ) interface need not be reentrant is added to the Description. A portable multi-threaded application should call *asctime_r*( ) instead of *asctime*( ).

asin( )

Arc sine function.

```
#include <math.h>

double asin(double x);
```

Issue 5: No functional changes in this issue.

asinh( )

Hyperbolic arc sine.

```
#include <math.h>
double asinh(double x);
```

Issue 5: No functional changes in this issue.

assert( )

Insert program diagnostics.

```
#include <assert.h>
void assert(int expression);
```

Issue 5: No functional changes in this issue.

atan( )

Arc tangent function.

```
#include <math.h>
double atan(double x);
```

Issue 5: No functional changes in this issue.

atan2( )

Arc tangent function.

```
#include <math.h>
double atan2(double y, double x);
```

Issue 5: No functional changes in this issue.

atanh( )

Hyperbolic arc tangent.

```
#include <math.h>
double atanh(double x);
```

Issue 5: No functional changes in this issue.

atexit( )

Register a function to run at process termination.

```
#include <stdlib.h>
int atexit(void (*func)(void));
```

Issue 5: No functional changes in this issue.

atof( )

Convert a string to a double-precision number.

```
#include <stdlib.h>
double atof(const char *str);
```

Issue 5: No functional changes in this issue.

atoi( )

Convert a string to an integer.

```
#include <stdlib.h>
int atoi(const char *str);
```

Issue 5: No functional changes in this issue.

atol( )

Convert a string to a long integer.

```
#include <stdlib.h>
long int atol(const char *str);
```

Issue 5: No functional changes in this issue.

basename( )

Return the last component of a pathname.

```
#include <libgen.h>
char *basename(char *path);
```

Issue 5: A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *basename*( ) when access to the function is serialized.

bcmp( )

Memory operations.

```
#include <strings.h>
int bcmp(const void *s1, const void *s2, size_t n);
```

Issue 5: No functional changes in this issue.

bcopy( )

Memory operations.

```
#include <strings.h>
void bcopy(const void *s1, void *s2, size_t n);
```

Issue 5: No functional changes in this issue.

brk, sbrk( )

Change space allocation. (**LEGACY**)

```
#include <unistd.h>

int brk(void *addr);
void *sbrk(intptr_t incr);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations. Applications are recommended to use *malloc*( ).

The type of the argument to *sbrk*( ) is changed from **int** to **intptr_t**. This change is for data size neutrality whereby systems supporting large address spaces may have problems representing appropriate values in a type **int**.

A note indicating that these interfaces need not be reentrant is added to the Description. A portable multi-threaded application may only safely call these functions when access to the function is serialized.

bsd_signal( )

Simplified signal facilities.

```
#include <signal.h>

void (*bsd_signal(int sig, void (*func)(int)))(int);
```

Issue 5: No functional changes in this issue.

bsearch( )

Binary search a sorted table.

```
#include <stdlib.h>

void *bsearch(const void *key, const void *base, size_t nel,
    size_t width, int (*compar)(const void *, const void *));
```

Issue 5: No functional changes in this issue.

btowc( )

Single-byte to wide-character conversion.

```
#include <stdio.h>
#include <wchar.h>

wint_t btowc(int c);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E). The *btowc*( ) function determines whether *c* constitutes a valid (one-byte) character in the initial shift state.

bzero( )

Memory operations.

```
#include <strings.h>
void bzero(void *s, size_t n);
```

Issue 5: No functional changes in this issue.

calloc( )

A memory allocator.

```
#include <stdlib.h>
void *calloc(size_t nelem, size_t elsize);
```

Issue 5: No functional changes in this issue.

catclose( )

Close a message catalog descriptor.

```
#include <nl_types.h>
int catclose(nl_catd catd);
```

Issue 5: No functional changes in this issue.

catgets( )

Read a program message.

```
#include <nl_types.h>
char *catgets(nl_catd catd, int set_id, int msg_id, const char *s);
```

Issue 5: A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *catgets*( ) when access to the function is serialized.

catopen( )

Open a message catalog.

```
#include <nl_types.h>
nl_catd catopen(const char *name, int oflag);
```

Issue 5: No functional changes in this issue.

cbrt( )

Cube root function.

```
#include <math.h>
double cbrt(double x);
```

Issue 5: No functional changes in this issue.

ceil( )

Ceiling value function.

```
#include <math.h>
double ceil(double x);
```

Issue 5: No functional changes in this issue.

cfgetispeed( )

Get input baud rate.

```
#include <termios.h>
speed_t cfgetispeed(const struct termios *termios_p);
```

Issue 5: No functional changes in this issue.

cfgetospeed( )

Get output baud rate.

```
#include <termios.h>
speed_t cfgetospeed(const struct termios *termios_p);
```

Issue 5: No functional changes in this issue.

cfsetispeed( )

Set input baud rate.

```
#include <termios.h>
int cfsetispeed(struct termios *termios_p, speed_t speed);
```

Issue 5: No functional changes in this issue.

cfsetospeed( )

Set output baud rate.

```
#include <termios.h>
int cfsetospeed(struct termios *termios_p, speed_t speed);
```

Issue 5: No functional changes in this issue.

chdir( )

Change working directory.

```
#include <unistd.h>
int chdir(const char *path);
```

Issue 5: No functional changes in this issue.

chmod( )

Change mode of a file.

```
#include <sys/types.h>
#include <sys/stat.h>

int chmod(const char *path, mode_t mode);
```

Issue 5: No functional changes in this issue.

chown( )

Change owner and group of a file.

```
#include <sys/types.h>
#include <unistd.h>

int chown(const char *path, uid_t owner, gid_t group);
```

Issue 5: No functional changes in this issue.

chroot( )

Change root directory. (**LEGACY**)

```
#include <unistd.h>

int chroot(const char *path);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations. Portable applications should not use this function.

A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *chroot*( ) when access to the function is serialized.

clearerr( )

Clear indicators on a stream.

```
#include <stdio.h>

void clearerr(FILE *stream);
```

Issue 5: No functional changes in this issue.

clock( )

Report CPU time used.

```
#include <time.h>

clock_t clock(void);
```

Issue 5: No functional changes in this issue.

clock_settime( ), clock_gettime( ), clock_getres( )

Clock and timer functions. (**REALTIME**)

```
#include <time.h>

int clock_settime(clockid_t clock_id, const struct timespec *tp);
int clock_gettime(clockid_t clock_id, struct timespec *tp);
int clock_getres(clockid_t clock_id, struct timespec *res);
```

Issue 5: These are new functions included for alignment with the POSIX Realtime Extension. These are part of the Realtime Feature Group and may not be available on all implementations.

The *clock_settime*( ) function sets the specified clock, *clock_id*, to the value specified by *tp*.

The *clock_gettime*( ) function returns the current value *tp* for the specified clock, *clock_id*.

The *clock_getres*( ) function returns the resolution of the specified clock, *clock_id*.

close( )

Close a file descriptor.

```
#include <unistd.h>

int close(int fildes);
```

Issue 5: The Description is updated for alignment with the POSIX Realtime Extension. Specifically, the semantics of *close*( ) are described for asynchronous or synchronous input/output, and also for memory mapped or shared memory objects.

The semantics of a *close*( ) function call when *fildes* is the master side of a pseudo-terminal are better described in this issue to match existing practice.

closedir( )

Close a directory stream.

```
#include <sys/types.h>
#include <dirent.h>

int closedir(DIR *dirp);
```

Issue 5: No functional changes in this issue.

closelog( ), openlog( ), setlogmask( ), syslog( )

Control system log.

```
#include <syslog.h>

void closelog(void);
void openlog(const char *ident, int logopt, int facility);
int setlogmask(int maskpri);
void syslog(int priority, const char *message, ... /* arguments */);
```

Issue 5: No functional changes in this issue.

compile( )

Produce a compiled regular expression. (**LEGACY**)

```
#include <regexp.h>

char *compile(char *instring, char *expbuf,
    const char *endbuf, int eof);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations. New applications should use the *regcomp*( ), *regexec*( ), *regerror*( ), and *regfree*( ) functions, and the **<regex.h>** header, which provide full internationalized regular expression functionality compatible with POSIX.2 and XBD, Chapter 7, *Regular Expressions*.

confstr( )

Get configurable variables.

```
#include <unistd.h>

size_t confstr(int name, char *buf, size_t len);
```

Issue 5: This function is changed to support new arguments which can be used to determine configuration strings for C compiler flags, linker/loader flags, and libraries for each different supported programming environment. This is related to the changes for data size neutrality.

The new arguments supported are:

_CS_XBS5_ILP32_OFF32_CFLAGS
_CS_XBS5_ILP32_OFF32_LDFLAGS
_CS_XBS5_ILP32_OFF32_LIBS
_CS_XBS5_ILP32_OFF32_LINTFLAGS
_CS_XBS5_ILP32_OFFBIG_CFLAGS
_CS_XBS5_ILP32_OFFBIG_LDFLAGS
_CS_XBS5_ILP32_OFFBIG_LIBS
_CS_XBS5_ILP32_OFFBIG_LINTFLAGS
_CS_XBS5_LP64_OFF64_CFLAGS
_CS_XBS5_LP64_OFF64_LDFLAGS
_CS_XBS5_LP64_OFF64_LIBS
_CS_XBS5_LP64_OFF64_LINTFLAGS
_CS_XBS5_LPBIG_OFFBIG_CFLAGS
_CS_XBS5_LPBIG_OFFBIG_LDFLAGS
_CS_XBS5_LPBIG_OFFBIG_LIBS
_CS_XBS5_LPBIG_OFFBIG_LINTFLAGS

cos( )

Cosine function.

```
#include <math.h>

double cos(double x);
```

Issue 5: No functional changes in this issue.

cosh( )

Hyperbolic cosine function.

```
#include <math.h>

double cosh(double x);
```

Issue 5: No functional changes in this issue.

creat( )

Create a new file or rewrite an existing one.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int creat(const char *path, mode_t mode);
```

Issue 5: No functional changes in this issue.

crypt( )

String encoding function. (**CRYPT**)

```
#include <unistd.h>

char *crypt (const char *key, const char *salt);
```

Issue 5: A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call this function when access to the function is serialized.

ctermid( )

Generate a pathname for controlling terminal.

```
#include <stdio.h>

char *ctermid(char *s);
```

Issue 5: The Description is updated for alignment with the POSIX Threads Extension. Specifically, if an application calling *ctermid*( ) uses any of the _POSIX_THREAD_SAFE_FUNCTIONS or _POSIX_THREADS interfaces, the *ctermid*( ) function must be called with a non-NULL parameter, else this function is not thread-safe.

ctime, ctime_r( )

Convert a time value to a date and time string.

```
#include <time.h>

char *ctime(const time_t *clock);
char *ctime_r(const time_t *clock, char *buf);
```

Issue 5: The *ctime_r*( ) function is a new function included for alignment with the POSIX Threads Extension. It is the thread-safe equivalent of the *ctime*( ) function.

A note indicating that the *ctime*( ) interface need not be reentrant is added to the Description. A portable multi-threaded application should call *ctime_r*( ) instead of *ctime*( ).

cuserid( )

Character login name of the user. (**LEGACY**)

```
#include <stdio.h>

char *cuserid(char *s);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations. This is due to differences between the functionality of *cuserid*( ) defined in IEEE Std 1003.1-1990 (POSIX.1), and that of historical implementations. In POSIX.1, *cuserid*( ) is removed completely.

Applications can obtain the XCU behavior by using:

```
getpwuid(getuid( ))
```

The historical behavior can be obtained by using:

```
getpwuid(geteuid( ))
```

The Description of this function is also updated for alignment with the POSIX Threads Extension. Specifically, if an application calling *cuserid*( ) uses any of the _POSIX_THREAD_SAFE_FUNCTIONS or _POSIX_THREADS interfaces, the *cuserid*( ) function must be called with a non-NULL parameter; otherwise, this function is not thread-safe.

A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *cuserid*( ) with a NULL parameter, when access to the function is serialized.

daylight( )

Daylight savings time flag.

```
#include <time.h>

extern int daylight;
```

Issue 5: No functional changes in this issue.

dbm_clearerr( ), dbm_close( ), dbm_delete( ), dbm_error( ), dbm_fetch( ), dbm_firstkey( ), dbm_nextkey( ), dbm_open( ), dbm_store( )

Database functions.

```
#include <ndbm.h>

int dbm_clearerr(DBM *db);
void dbm_close(DBM *db);
int dbm_delete(DBM *db, datum key);
int dbm_error(DBM *db);
datum dbm_fetch(DBM *db, datum key);
datum dbm_firstkey(DBM *db);
datum dbm_nextkey(DBM *db);
DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
int dbm_store(DBM *db, datum key, datum content, int store_mode);
```

Issue 5: A note indicating that these interfaces need not be reentrant is added to the Description. A portable multi-threaded application may only safely call these functions when access to the function is serialized.

The minimum key/content pair is changed from 1024 to 1023 bytes.

The semantics of the *open_flags* argument when the O_APPEND flag is set are now explicitly described as unspecified.

A new paragraph is added:

> ‘‘The sum of the sizes of a key/content pair must not exceed the internal block size. Moreover, all key/content pairs that hash together must fit on a single block. The *dbm_store*( ) function returns an error in the event that a disk block fills with inseparable data.’’

difftime( )

Compute the difference between two calendar time values.

```
#include <time.h>

double difftime(time_t time1, time_t time0);
```

Issue 5: No functional changes in this issue.

dirname( )

Report the parent directory name of a file pathname.

```
#include <libgen.h>

char *dirname(char *path);
```

Issue 5: A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *dirname*( ) when access to the function is serialized.

div( )

Compute the quotient and remainder of an integer division.

```
#include <stdlib.h>

div_t div(int numer, int denom);
```

Issue 5: No functional changes in this issue.

dlclose( )

Close a *dlopen*( ) object.

```
#include <dlfcn.h>

int dlclose(void *handle);
```

Issue 5: This is a new function added to support dynamic linking.

dlclose( ) is used to inform the system that the object referenced by a *handle* returned from a previous *dlopen*( ) invocation is no longer needed by the application.

The use of *dlclose*( ) reflects a statement of intent on the part of the process, but does not create any requirement upon the implementation, such as removal of the code or symbols referenced by *handle*.

For example:

```
...
/* open a shared object then close it ... */

#include <dlfcn.h>
void *mylib;
int foo;

mylib = dlopen ("mylib.so.1", RTLD_LAZY);

foo = dlcose( mylib );

...
```

dlerror( )

Get diagnostic information.

```
#include <dlfcn.h>

char *dlerror(void);
```

Issue 5: This is a new function added to support dynamic linking. This function is used to find out diagnostic information if any of the other dynamic linking functions (*dlopen*( ), *dlclose*( ), and *dlsym*( )) returned an error. A call to *dlerror*( ) returns a null-terminated character string (with no trailing newline) that describes the last error that occurred during dynamic linking processing. If no dynamic linking errors have occurred since the last invocation of *dlerror*( ), *dlerror*( ) returns NULL.

For example:

```
...
#include <dlfcn.h>

char *errstr

errstr = dlerror();
if (errstr != NULL )
    printf ("A dynamic linking error occurred: (%s)\n", errstr);
...
```

dlopen( )

Gain access to an executable object file.

```
#include <dlfcn.h>

void *dlopen(const char *file, int mode);
```

Issue 5: This is a new function added to support dynamic linking.

dlopen( ) makes an executable object file specified by *file* available to the calling program. The class of files eligible for this operation and the manner of their construction are specified by the implementation, although typically such files are executable objects, such as shared libraries, relocatable files, or programs. For example:

```
...
/* Open the shared object mylib.so.1 for Lazy binding */

#include <dlfcn.h>
void *mylib;

mylib = dlopen ("mylib.so.1", RTLD_LAZY);
...
```

dlsym()

Obtain the address of a symbol from a *dlopen*( ) object.

```
#include <dlfcn.h>

void *dlsym(void *handle, const char *name);
```

Issue 5: This is a new function added to support dynamic linking.

dlsym( ) allows a process to obtain the address of a symbol defined within an object made accessible through a *dlopen*( ) call. *handle* is the value returned from a call to *dlopen*( ) (and which has not since been released via a call to *dlclose*( )). *name* is the symbol's name as a character string.

For example:

```
...
#include <dlfcn.h>
void *mylib;
int (*myptr) ();

mylib = dlopen ("mylib.so.1", RTLD_LAZY);

myptr = (int (*) ()) dlsym (mylib, "mysymname");
...
```

drand48(), erand48(), jrand48(), lcong48(), lrand48(), mrand48(), nrand48(), seed48(), srand48()

Generate uniformly distributed pseudo-random numbers.

```
#include <stdlib.h>

double drand48(void);
double erand48(unsigned short int xsubi[3]);
long int jrand48(unsigned short int xsubi[3]);
void lcong48(unsigned short int param[7]);
long int lrand48(void);
long int mrand48(void);
long int nrand48(unsigned short int xsubi[3]);
unsigned short int *seed48(unsigned short int seed16v[3]);
void srand48(long int seedval);
```

Issue 5: A note indicating that these interfaces need not be reentrant is added to the Description. A portable multi-threaded application may only safely call these functions when access to the function is serialized.

dup( ), dup2( )

Duplicate an open file descriptor.

```
#include <unistd.h>

int dup(int fildes);
int dup2(int fildes, int fildes2);
```

Issue 5: No functional changes in this issue.

ecvt( ), fcvt( ), gcvt( )

Convert a floating-point number to a string.

```
#include <stdlib.h>

char *ecvt(double value, int ndigit, int *decpt, int *sign);
char *fcvt(double value, int ndigit, int *decpt, int *sign);
char *gcvt(double value, int ndigit, char *buf);
```

Issue 5: A note indicating that these interfaces need not be reentrant is added to the Description. A portable multi-threaded application may only safely call these functions when access to the function is serialized.

encrypt( )

Encoding function. (**CRYPT**)

```
#include <unistd.h>

void encrypt(char block[64], int edflag);
```

Issue 5: A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *encrypt*( ) when access to the function is serialized.

endgrent( ), getgrent( ), setgrent( )

Group database entry functions.

```
#include <grp.h>

void endgrent(void);
struct group *getgrent(void);
void setgrent(void);
```

Issue 5: A note indicating that these interfaces need not be reentrant is added to the Description. A portable multi-threaded application may only safely call these functions when access to the function is serialized.

endpwent( ), getpwent( ), setpwent( )

User database functions.

```
#include <pwd.h>

void endpwent(void);
struct passwd *getpwent(void);
void setpwent(void);
```

Issue 5: A note indicating that these interfaces need not be reentrant is added to the Description. A portable multi-threaded application may only safely call these functions when access to the function is serialized.

endutxent( ), getutxent( ), getutxid( ), getutxline( ), pututxline( ), setutxent( )

User accounting database functions.

```
#include <utmpx.h>

void endutxent(void);
struct utmpx *getutxent(void);
struct utmpx *getutxid(const struct utmpx *id);
struct utmpx *getutxline(const struct utmpx *line);
struct utmpx *pututxline(const struct utmpx *utmpx);
void setutxent(void);
```

Issue 5: A note indicating that these interfaces need not be reentrant is added to the Description. A portable multi-threaded application may only safely call these functions when access to the function is serialized.

environ( )

Array of character pointers to the environment strings.

```
extern char **environ;
```

Issue 5: No functional changes in this issue.

erand48( )

Generate uniformly distributed pseudo-random numbers.

```
#include <stdlib.h>

double erand48(unsigned short int xsubi[3]);
```

Issue 5: No functional changes in this issue.

erf( ), erfc( )

Error and complementary error functions.

```
#include <math.h>

double erf(double x);
double erfc(double x);
```

Issue 5: No functional changes in this issue.

errno( )

XSI error return value.

```
#include <errno.h>
```

Issue 5: The following sentence is deleted from the Description: ‘‘The value of *errno* is 0 at program startup, but is never set to 0 by any XSI function.’’ The Description also no longer states that conforming implementations may support the declaration:

```
extern int errno;
```

Both these historical behaviors are obsolete and may not be supported by some implementations. Applications should include the **<errno.h>** header to receive the appropriate definition of *errno*.

environ( ), execl( ), execv( ), execle( ), execve( ), execlp( ), execvp( )

Execute a file.

```
#include <unistd.h>

extern char **environ;
int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
int execv(const char *path, char *const argv[]);
int execle(const char *path,
    const char *arg0, ... /*, (char *)0, char *const envp[]*/);
int execve(const char *path, char *const argv[], char *const envp[]);
int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
int execvp(const char *file, char *const argv[]);
```

Issue 5: The Description is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension. Specifically, the following changes are made:

- Conforming multi-threaded applications will not use the *environ* variable to access or modify any environment variable while any other thread is concurrently modifying any environment variable.
- A call to any *exec*( ) function from a process with more than one thread results in all threads being terminated and the new executable image being loaded and executed.
- Semantics for named semaphores, memory locking, process scheduling, memory mappings, timers, message passing, and asynchronous input/output are described.
- Large File Support extensions are added; see Chapter 17 on page 163.

exit( ), _exit( )

Terminate a process.

```
#include <stdlib.h>

void exit(int status);

#include <unistd.h>

void _exit(int status);
```

Issue 5: The Description is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension. Specifically, semantics are described for named semaphores, memory locking, process scheduling, memory mappings, message passing, and asynchronous input/output.

Threads terminated by a call to *_exit*( ) will not invoke their cancelation cleanup handlers or per-thread data destructors.

Interactions with the SA_NOCLDWAIT flag and SIGCHLD signal are further clarified.

The values of *status* from *exit*( ) are better described.

exp( )

Exponential function.

```
#include <math.h>

double exp(double x);
```

Issue 5: No functional changes in this issue.

expm1( )

Compute exponential functions.

```
#include <math.h>

double expm1 (double x);
```

Issue 5: No functional changes in this issue.

fabs( )

Absolute value function.

```
#include <math.h>

double fabs(double x);
```

Issue 5: No functional changes in this issue.

fattach( )

Attach a STREAMS-based file descriptor to a file in the filesystem namespace.

```
#include <stropts.h>

int fattach(int fildes, const char *path);
```

Issue 5: The [EXDEV] error is added to the list of optional errors for the case when a link to a file on another filesystem was attempted. This reflects historical practice of many systems that implement *fattach*( ) in a similar manner to the traditional *mount*( ) function.

fchdir( )

Change working directory.

```
#include <unistd.h>

int fchdir(int fildes);
```

Issue 5: No functional changes in this issue.

fchmod( )

Change mode of a file.

```
#include <sys/stat.h>

int fchmod(int fildes, mode_t mode);
```

Issue 5: Aligned with *fchmod*( ) in the POSIX Realtime Extension. Specifically, the second paragraph of the Description is added, describing the semantics for operations on Shared Memory Objects, and a second instance of [EINVAL] is defined in the list

of optional errors. The Realtime semantics may only be supported on implementations supporting the Realtime Feature Group.

fchown( )

Change owner and group of a file.

```
#include <unistd.h>

int fchown(int fildes, uid_t owner, gid_t group);
```

Issue 5: No functional changes in this issue.

fclose( )

Close a stream.

```
#include <stdio.h>

int fclose(FILE *stream);
```

Issue 5: Large File Support extensions are added; see Chapter 17 on page 163.

fcntl( )

File control.

```
#include <fcntl.h>

int fcntl(int fildes, int cmd, ...);
```

Issue 5: The Description is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension, specifically the semantics for shared memory objects are described.

Large File Support extensions are added; see Chapter 17 on page 163.

fcvt( )

Convert a floating-point number to a string.

```
#include <stdlib.h>

char *fcvt(double value, int ndigit, int *decpt, int *sign);
```

Issue 5: No functional changes in this issue.

FD_CLR( )

Macros for synchronous I/O multiplexing.

```
#include <sys/time.h>

FD_CLR(int fd, fd_set *fdset);
FD_ISSET(int fd, fd_set *fdset);
FD_SET(int fd, fd_set *fdset);
FD_ZERO(fd_set *fdset);
```

Issue 5: No functional changes in this issue.

fdatasync( )

Synchronize the data of a file. (**REALTIME**)

```
#include <unistd.h>

int fdatasync(int fildes);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *fdatasync*( ) function forces all currently queued I/O operations associated with the file indicated by file descriptor *fildes* to the synchronized I/O completion state.

The functionality is as per *fsync*( ) (with the symbol _XOPEN_REALTIME defined), with the exception that all I/O operations are completed as defined for synchronized I/O data integrity completion.

fdetach( )

Detach a name from a STREAMS-based file descriptor.

```
#include <stropts.h>

int fdetach(const char *path);
```

Issue 5: No functional changes in this issue.

fdopen( )

Associate a stream with a file descriptor.

```
#include <stdio.h>

FILE *fdopen(int fildes, const char *mode);
```

Issue 5: The Description is updated for alignment with the POSIX Realtime Extension. Specifically, the results of the *fdopen*( ) function on a shared memory object is unspecified.

Large File Support extensions are added; see Chapter 17 on page 163.

feof( )

Test end-of-file indicator on a stream.

```
#include <stdio.h>

int feof(FILE *stream);
```

Issue 5: No functional changes in this issue.

ferror( )

Test error indicator on a stream.

```
#include <stdio.h>
int ferror(FILE *stream);
```

Issue 5: No functional changes in this issue.

fflush( )

Flush a stream.

```
#include <stdio.h>
int fflush(FILE *stream);
```

Issue 5: Large File Support extensions are added; see Chapter 17 on page 163.

ffs( )

Find first set bit.

```
#include <strings.h>
int ffs(int i);
```

Issue 5: No functional changes in this issue.

fgetc( )

Get a byte from a stream.

```
#include <stdio.h>
int fgetc(FILE *stream);
```

Issue 5: Large File Support extensions are added; see Chapter 17 on page 163.

fgetpos( )

Get current file position information.

```
#include <stdio.h>
int fgetpos(FILE *stream, fpos_t *pos);
```

Issue 5: Large File Support extensions are added; see Chapter 17 on page 163.

fgets( )

Get a string from a stream.

```
#include <stdio.h>
char *fgets(char *s, int n, FILE *stream);
```

Issue 5: No functional changes in this issue.

fgetwc( )

Get a wide-character code from a stream.

```
#include <stdio.h>
#include <wchar.h>

wint_t fgetwc(FILE *stream);
```

Issue 5: The Optional Header (OH) marking is removed from **<stdio.h>**.

Large File Support extensions are added; see Chapter 17 on page 163.

fgetws( )

Get a wide-character string from a stream.

```
#include <stdio.h>
#include <wchar.h>

wchar_t *fgetws(wchar_t *ws, int n, FILE *stream);
```

Issue 5: The Optional Header (OH) marking is removed from **<stdio.h>**.

fileno( )

Map a stream pointer to a file descriptor.

```
#include <stdio.h>

int fileno(FILE *stream);
```

Issue 5: No functional changes in this issue.

flockfile( ), ftrylockfile( ), funlockfile( )

Stdio locking functions.

```
#include <stdio.h>

void flockfile(FILE *file);
int ftrylockfile(FILE *file);
void funlockfile(FILE *file);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The *flockfile*( ), *ftrylockfile*( ), and *funlockfile*( ) functions provide for explicit application-level locking of *stdio*(**FILE***) objects.

floor( )

Floor function.

```
#include <math.h>

double floor(double x);
```

Issue 5: No functional changes in this issue.

fmod( )

Floating-point remainder value function.

```
#include <math.h>

double fmod(double x, double y);
```

Issue 5: No functional changes in this issue.

fmtmsg( )

Display a message in the specified format on standard error and/or a system console.

```
#include <fmtmsg.h>

int fmtmsg(long classification, const char *label, int severity,
    const char *text, const char *action, const char *tag);
```

Issue 5: No functional changes in this issue.

fnmatch( )

Match a filename or a pathname.

```
#include <fnmatch.h>

int fnmatch(const char *pattern, const char *string, int flags);
```

Issue 5: No functional changes in this issue.

fopen( )

Open a stream.

```
#include <stdio.h>

FILE *fopen(const char *filename, const char *mode);
```

Issue 5: Large File Support extensions are added; see Chapter 17 on page 163.

fork( )

Create a new process.

```
#include <sys/types.h>
#include <unistd.h>

pid_t fork(void);
```

Issue 5: The Description is changed for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Semantics for named semaphores, memory locking, process scheduling, memory mappings, timers, message passing, and asynchronous input/output are described.

A process is created with a single thread. If a multi-threaded process calls *fork*( ), the new process contains a replica of the calling thread and its entire address space, possibly including the states of mutexes and other resources.

fpathconf( ), pathconf( )

Get configurable pathname variables.

```
#include <unistd.h>

long int fpathconf(int fildes, int name);
long int pathconf(const char *path, int name);
```

Issue 5: The Description is updated for alignment with the POSIX Realtime Extension. Specifically, the variables _POSIX_ASYNC_IO, _POSIX_PRIO_IO, and _POSIX_SYNC_IO, and the *name* values _PC_ASYNC_IO, _PC_PRIO_IO, and _PC_SYNC_IO are added.

Large File Support extensions are added; see Chapter 17 on page 163.

fprintf( ), printf( ), snprintf( ), sprintf( )

Print formatted output.

```
#include <stdio.h>

int fprintf(FILE *stream, const char *format, ...);
int printf(const char *format, ...);
int snprintf(char *s, size_t n, const char *format, ...);
int sprintf(char *s, const char *format, ...);
```

Issue 5: These functions are aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the *l* (ell) qualifier can now be used with *c* and *s* conversion characters.

The *snprintf*( ) function is new. *snprintf*( ) is identical to *sprintf*( ) with the addition of the *n* argument, which states the size of the buffer referred to by *s*.

fputc( )

Put a byte on a stream.

```
#include <stdio.h>

int fputc(int c, FILE *stream);
```

Issue 5: Large File Support extensions are added; see Chapter 17 on page 163.

fputs( )

Put a string on a stream.

```
#include <stdio.h>

int fputs(const char *s, FILE *stream);
```

Issue 5: No functional changes in this issue.

fputwc( )

Put a wide-character code on a stream.

```
#include <stdio.h>
#include <wchar.h>

wint_t fputwc(wchar_t wc, FILE *stream);
```

Issue 5: Aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc* is changed from **wint_t** to **wchar_t**.

The Optional Header (OH) marking is removed from **<stdio.h>**.

Large File Support extensions are added; see Chapter 17 on page 163.

fputws( )

Put a wide-character string on a stream.

```
#include <stdio.h>
#include <wchar.h>

int fputws(const wchar_t *ws, FILE *stream);
```

Issue 5: The Optional Header (OH) marking is removed from **<stdio.h>**.

fread( )

Binary input.

```
#include <stdio.h>

size_t fread(void *ptr, size_t size, size_t nitems, FILE *stream);
```

Issue 5: No functional changes in this issue.

free( )

Free allocated memory.

```
#include <stdlib.h>

void free(void *ptr);
```

Issue 5: No functional changes in this issue.

freopen( )

Open a stream.

```
#include <stdio.h>

FILE *freopen(const char *filename, const char *mode, FILE *stream);
```

Issue 5: The Description is updated to indicate that the orientation of the stream is cleared and the conversion state of the stream is set to an initial conversion state by a successful call to the *freopen*( ) function.

Large File Support extensions are added; see Chapter 17 on page 163.

frexp( )

Extract mantissa and exponent from a double precision number.

```
#include <math.h>

double frexp(double num, int *exp);
```

Issue 5: No functional changes in this issue.

fscanf( ), scanf( ), sscanf( )

Convert formatted input.

```
#include <stdio.h>

int fscanf(FILE *stream, const char *format, ...);
int scanf(const char *format, ...);
int sscanf(const char *s, const char *format, ...);
```

Issue 5: Aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the *l* (ell) qualifier is now defined for *c*, *s*, and [conversion characters.

The Description is updated to indicate that if infinity and NaN can be generated by the *fprintf*( ) family of functions, then they will be recognized by the *fscanf*( ) family.

fseek( ), fseeko( )

Reposition a file-position indicator in a stream.

```
#include <stdio.h>

int fseek(FILE *stream, long int offset, int whence);
int fseeko(FILE *stream, off_t offset, int whence);
```

Issue 5: Large File Support extensions are added; see Chapter 17 on page 163. Specifically, the *fseeko*( ) function is new.

fsetpos( )

Set current file position.

```
#include <stdio.h>

int fsetpos(FILE *stream, const fpos_t *pos);
```

Issue 5: No functional changes in this issue.

fstat( )

Get file status.

```
#include <sys/types.h>
#include <sys/stat.h>

int fstat(int fildes, struct stat *buf);
```

Issue 5: The Description is updated for alignment with the POSIX Realtime Extension. Specifically, the semantics for a shared memory object are described.

Large File Support extensions are added; see Chapter 17 on page 163.

fstatvfs(), statvfs()

Get filesystem information.

```
#include <sys/statvfs.h>

int fstatvfs(int fildes, struct statvfs *buf);
int statvfs(const char *path, struct statvfs *buf);
```

Issue 5: Large File Support extensions are added; see Chapter 17 on page 163.

fsync()

Synchronize changes to a file.

```
#include <unistd.h>

int fsync(int fildes);
```

Issue 5: Aligned with *fsync*() in the POSIX Realtime Extension. Specifically, the Description and Return Value sections are much expanded, and the Errors section is updated to indicate that *fsync*() can return the error conditions defined for *read*() and *write*().

ftell(), ftello()

Return a file offset in a stream.

```
#include <stdio.h>

long int ftell(FILE *stream);
off_t ftello(FILE *stream);
```

Issue 5: Large File Support extensions are added; see Chapter 17 on page 163.

ftime()

Get date and time.

```
#include <sys/timeb.h>

int ftime(struct timeb *tp);
```

Issue 5: No functional changes in this issue.

ftok()

Generate an IPC key.

```
#include <sys/ipc.h>

key_t ftok(const char *path, int id);
```

Issue 5: No functional changes in this issue.

ftruncate( ), truncate( )

Truncate a file to a specified length.

```
#include <unistd.h>

int ftruncate(int fildes, off_t length);
int truncate(const char *path, off_t length);
```

Issue 5: Aligned with *ftruncate*( ) in the POSIX Realtime Extension. Specifically, the Description is extensively reworded and [EROFS] is added to the list of mandatory errors that can be returned by *ftruncate*( ).

Large File Support extensions are added; see Chapter 17 on page 163.

ftrylockfile( )

Stdio locking functions.

```
#include <stdio.h>

int ftrylockfile(FILE *file);
```

Issue 5: This is a new function included for alignment with the POSIX Threads Extension.

ftw( )

Traverse (walk) a file tree.

```
#include <ftw.h>

int ftw(const char *path, int (*fn)(const char *,
    const struct stat *ptr, int flag), int ndirs);
```

Issue 5: No functional changes in this issue.

funlockfile( )

Stdio locking functions.

```
#include <stdio.h>

void funlockfile(FILE *file);
```

Issue 5: This is a new function included for alignment with the POSIX Threads Extension.

fwide( )

Set stream orientation.

```
#include <stdio.h>
#include <wchar.h>

int fwide(FILE *stream, int mode);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The *fwide*( ) function determines the orientation, byte-oriented or wide-oriented, of the stream pointed to by *stream*.

fwprintf( ), wprintf( ), swprintf( )

Print formatted wide-character output.

```
#include <stdio.h>
#include <wchar.h>

int fwprintf(FILE *stream, const wchar_t *format, ...);
int wprintf(const wchar_t *format, ...);
int swprintf(wchar_t *s, size_t n, const wchar_t *format, ...);
```

Issue 5: Included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E). These are wide-character versions of the *fprintf*( ), *printf*( ), and *sprintf*( ) functions.

fwrite( )

Binary output.

```
#include <stdio.h>

size_t fwrite(const void *ptr, size_t size, size_t nitems,
    FILE *stream);
```

Issue 5: No functional changes in this issue.

fwscanf( ), wscanf( ), swscanf( )

Convert formatted wide-character input.

```
#include <stdio.h>
#include <wchar.h>

int fwscanf(FILE *stream, const wchar_t *format, ...);
int wscanf(const wchar_t *format, ...);
int swscanf(const wchar_t *s, const wchar_t *format, ...);
```

Issue 5: These are new functions included for alignment with the the ISO/IEC 9899:1990/Amendment 1:1995 (E). These are wide-character versions of the *fscanf*( ), *scanf*( ), and *sscanf*( ) functions.

gamma( ), signgam( )

Log gamma function. (**LEGACY**)

```
#include <math.h>

double gamma(double x);
extern int signgam;
```

Issue 5: A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *gamma*( ) when access to the function is serialized.

This function is marked LEGACY and may not be available on all implementations.

gcvt( )

Convert a floating-point number to a string.

```
#include <stdlib.h>

char *gcvt(double value, int ndigit, char *buf);
```

Issue 5: No functional changes in this issue.

getc( )

Get a byte from a stream.

```
#include <stdio.h>

int getc(FILE *stream);
```

Issue 5: No functional changes in this issue.

getchar( )

Get a byte from a *stdin* stream.

```
#include <stdio.h>

int getchar(void);
```

Issue 5: No functional changes in this issue.

getc_unlocked( ), getchar_unlocked( ), putc_unlocked( ), putchar_unlocked( )

Stdio with explicit client locking.

```
#include <stdio.h>

int getc_unlocked(FILE *stream);
int getchar_unlocked(void);
int putc_unlocked(int c, FILE *stream);
int putchar_unlocked(int c);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

These are functionally identical to *getc*( ), *getchar*( ), *putc*( ), and *putchar*( ) with the exception that they are not required to be implemented in a thread-safe manner. They may only safely be used within a scope protected by *flockfile*( ) (or *ftrylockfile*( )) and *funlockfile*( ).

These functions may safely be used in a multi-threaded program if and only if they are called while the invoking thread owns the (**FILE***) object.

getcontext( ), setcontext( )

Get and set current user context.

```
#include <ucontext.h>

int getcontext(ucontext_t *ucp);
int setcontext(const ucontext_t *ucp);
```

Issue 5: The following sentence was removed from the Description:

''If the *ucp* argument was passed to a signal handler, program execution continues with the program instruction following the instruction interrupted by the signal.''

getcwd( )

Get the pathname of the current working directory.

```
#include <unistd.h>

char *getcwd(char *buf, size_t size);
```

Issue 5: No functional changes in this issue.

getdate( )

Convert user format date and time.

```
#include <time.h>

struct tm *getdate(const char *string);
```

Issue 5: The last paragraph of the Description is added, encouraging applications to remove declarations of *getdate_err* and instead incorporate the declaration by including **<time.h>**.

The %*C* specifier is added, and the exact meaning of the %*y* specifier is clarified in the Description.

%*C* Century number (00-99; leading zeros are permitted but not required)

%*y* Year within century. When a century is not otherwise specified, values in the range 69-99 refer to years in the twentieth century (1969 to 1999 inclusive); values in the range 00-68 refer to years in the twenty-first century (2000 to 2068 inclusive).

Portable applications are recommended to use the %*Y* specifier (4-digit years) rather than %*y* (2-digit years), to avoid problems with the millennium rollover.

A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *getdate*( ) when access to the function is serialized.

The %*R* specifier is changed to follow historical practice.

getdtablesize( )

Get the file descriptor table size. (**LEGACY**)

```
#include <unistd.h>

int getdtablesize(void);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations. Applications should use the *getrlimit*( ) function instead.

A new paragraph is added to the Application Usage section giving reasons why the interface may be withdrawn in a future issue. Specifically, the change is made for data size neutrality. The *getdtablesize*( ) function returns the size of the file descriptor table. This is equivalent to *getrlimit*( ) with the RLIMIT_NOFILE option. Whereas the *getrlimit*( ) function returns a value of type **rlim_t**. This function,

returning an **int**, may have problems representing appropriate values in the future.

A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *getdtablesize*( ) when access to the function is serialized.

getegid( )

Get the effective group ID.

```
#include <sys/types.h>
#include <unistd.h>

gid_t getegid(void);
```

Issue 5: No functional changes in this issue.

getenv( )

Get value of an environment variable.

```
#include <stdlib.h>

char *getenv(const char *name);
```

Issue 5: A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *getenv*( ) when access to the function is serialized.

geteuid( )

Get the effective user ID.

```
#include <sys/types.h>
#include <unistd.h>

uid_t geteuid(void);
```

Issue 5: No functional changes in this issue.

getgid( )

Get the real group ID.

```
#include <sys/types.h>
#include <unistd.h>

gid_t getgid(void);
```

Issue 5: No functional changes in this issue.

getgrent( )

Get the group database entry.

```
#include <grp.h>

struct group *getgrent(void);
```

Issue 5: No functional changes in this issue.

getgrgid(), getgrgid_r()

Get group database entry for a group ID.

```
#include <grp.h>

struct group *getgrgid(gid_t gid);
int getgrgid_r(gid_t gid, struct group *grp, char *buffer,
    size_t bufsize, struct group **result);
```

Issue 5: The *getgrgid_r*() function is included for alignment with the POSIX Threads Extension, and is the thread-safe equivalent of *getgrgid*().

A note indicating that the *getgrgid*() interface need not be reentrant is added to the Description. A portable multi-threaded application should call *getgrid_r*() instead of *getgrid*().

getgrnam(), getgrnam_r()

Search group database for a name.

```
#include <sys/types.h>
#include <grp.h>

struct group *getgrnam(const char *name);
int getgrnam_r(const char *name, struct group *grp, char *buffer,
    size_t bufsize, struct group **result);
```

Issue 5: The *getgrnam_r*() function is included for alignment with the POSIX Threads Extension, and is the thread-safe equivalent of *getgrnam*().

A note indicating that the *getgrnam*() interface need not be reentrant is added to the Description. A portable multi-threaded application should call *getgrnam_r*() instead of *getgrnam*().

getgroups()

Get supplementary group IDs.

```
#include <sys/types.h>
#include <unistd.h>

int getgroups(int gidsetsize, gid_t grouplist[]);
```

Issue 5: No functional changes in this issue.

gethostid()

Get an identifier for the current host.

```
#include <unistd.h>

long gethostid(void);
```

Issue 5: No functional changes in this issue.

getitimer( ), setitimer( )

Get or set value of interval timer.

```
#include <sys/time.h>

int getitimer(int which, struct itimerval *value);
int setitimer(int which, const struct itimerval *value,
    struct itimerval *ovalue);
```

Issue 5: No functional changes in this issue.

getlogin( ), getlogin_r( )

Get login name.

```
#include <unistd.h>

char *getlogin(void);
int getlogin_r(char *name, size_t namesize);
```

Issue 5: The *getlogin_r*( ) function is included for alignment with the POSIX Threads Extension, and is the thread-safe equivalent of *getlogin*( ).

A note indicating that the *getlogin*( ) interface need not be reentrant is added to the Description. A portable multi-threaded application should call *getlogin_r*( ) instead of *getlogin*( ).

getmsg( ), getpmsg( )

Receive next message from a STREAMS file.

```
#include <stropts.h>

int getmsg(int fildes, struct strbuf *ctlptr, struct strbuf *dataptr,
    int *flagsp);
int getpmsg(int fildes, struct strbuf *ctlptr, struct strbuf *dataptr,
    int *bandp, int *flagsp);
```

Issue 5: A paragraph regarding ‘‘high-priority control parts of messages’’ is added to the Return Value section.

getopt( ), optarg( ), optind( ), opterr( ), optopt( )

Command option parsing.

```
#include <unistd.h>

int getopt(int argc, char * const argv[], const char *optstring);
extern char *optarg;
extern int optind, opterr, optopt;
```

Issue 5: A note indicating that the *getopt*( ) interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *getopt*( ) when access to the function is serialized.

getpagesize( )

Get the current page size. (**LEGACY**)

```
#include <unistd.h>

int getpagesize(void);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations. Portable applications should call *sysconf*( ). The *getpagesize*( ) function is equivalent to *sysconf*(_SC_PAGE_SIZE) and *sysconf*(_SC_PAGESIZE).

A new paragraph is added to the Application Usage section indicating why the interface may be withdrawn in a future issue. This is a change made for data size neutrality. This function, returning an **int**, may have problems representing appropriate values in the future. Also, the behavior is not specified for this function on systems that support variable size pages. On variable page size systems, a page can be extremely large (theoretically, up to the size of memory). This allows very efficient address translations for large segments of memory that have common page attributes.

A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *getpagesize*( ) when access to the function is serialized.

getpass( )

Read a string of characters without echo. (**LEGACY**)

```
#include <unistd.h>

char *getpass(const char *prompt);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations.

A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *getpass*( ) when access to the function is serialized.

getpgid( )

Get the process group ID for a process.

```
#include <unistd.h>

pid_t getpgid(pid_t pid);
```

Issue 5: No functional changes in this issue.

getpgrp( )

Get the process group ID of the calling process.

```
#include <sys/types.h>
#include <unistd.h>

pid_t getpgrp(void);
```

Issue 5: No functional changes in this issue.

getpid( )

Get the process ID.

```
#include <sys/types.h>
#include <unistd.h>

pid_t getpid(void);
```

Issue 5: No functional changes in this issue.

getpmsg( )

Get the user database entry.

```
#include <pwd.h>

int getpmsg(int fildes, struct strbuf *ctlptr, struct strbuf *dataptr,
    int *bandp, int *flagsp);
```

Issue 5: No functional changes in this issue.

getppid( )

Get the parent process ID.

```
#include <sys/types.h>
#include <unistd.h>

pid_t getppid(void);
```

Issue 5: No functional changes in this issue.

getpriority( ), setpriority( )

Get or set the nice value.

```
#include <sys/resource.h>

int getpriority(int which, id_t who);
int setpriority(int which, id_t who, int value);
```

Issue 5: The Description is reworded in terms of the *nice* value rather than *priority* to avoid confusion with functionality in the POSIX Realtime Extension.

getpwent( )

Get user database entry.

```
#include <pwd.h>

struct passwd *getpwent(void);
```

Issue 5: No functional changes in this issue.

getpwnam(), getpwnam_r()

Search user database for a name.

```
#include <sys/types.h>
#include <pwd.h>

struct passwd *getpwnam(const char *name);
int getpwnam_r(const char *nam, struct passwd *pwd, char *buffer,
    size_t bufsize, struct passwd **result);
```

Issue 5: This describes the return value as possibly being a pointer to static data, and how an application should check for an error.

The *getpwnam_r*() function is included for alignment with the POSIX Threads Extension, and is the thread-safe equivalent of *getpwnam*().

A note indicating that the *getpwnam*() interface need not be reentrant is added to the Description. A portable multi-threaded application should call *getpwnam_r*() instead of *getpwnam*().

getpwuid(), getpwuid_r()

Search user database for a user ID.

```
#include <sys/types.h>
#include <pwd.h>

struct passwd *getpwuid(uid_t uid);
int getpwuid_r(uid_t uid, struct passwd *pwd, char *buffer,
    size_t bufsize, struct passwd **result);
```

Issue 5: The *getpwuid_r*() function is included for alignment with the POSIX Threads Extension, and is the thread-safe equivalent of *getpwuid*().

A note indicating that the *getpwuid*() interface need not be reentrant is added to the Description. A portable multi-threaded application should call *getpwuid_r*() instead of *getpwuid*().

getrlimit(), setrlimit()

Control maximum resource consumption.

```
#include <sys/resource.h>

int getrlimit(int resource, struct rlimit *rlp);
int setrlimit(int resource, const struct rlimit *rlp);
```

Issue 5: An Application Usage section is added.

Large File Support extensions are added; see Chapter 17 on page 163.

getrusage( )

Get information about resource utilization.

```
#include <sys/resource.h>
int getrusage(int who, struct rusage *r_usage);
```

Issue 5: No functional changes in this issue.

gets( )

Get a string from a *stdin* stream.

```
#include <stdio.h>
char *gets(char *s);
```

Issue 5: No functional changes in this issue.

getsid( )

Get the process group ID of the session leader.

```
#include <unistd.h>
pid_t getsid(pid_t pid);
```

Issue 5: No functional changes in this issue.

getsubopt( )

Parse suboption arguments from a string.

```
#include <stdlib.h>
int getsubopt(char **optionp, char * const *tokens, char **valuep);
```

Issue 5: No functional changes in this issue.

gettimeofday( )

Get the date and time.

```
#include <sys/time.h>
int gettimeofday(struct timeval *tp, void *tzp);
```

Issue 5: No functional changes in this issue.

getuid( )

Get a real user ID.

```
#include <sys/types.h>
#include <unistd.h>
uid_t getuid (void);
```

Issue 5: No functional changes in this issue.

getutxent(), getutxid, getutxline()

Get user accounting database entries.

```
#include <utmpx.h>

struct utmpx *getutxent(void);
struct utmpx *getutxid(const struct utmpx *id);
struct utmpx *getutxline(const struct utmpx *line);
```

Issue 5: No functional changes in this issue.

getw()

Get a word from a stream. (**LEGACY**)

```
#include <stdio.h>

int getw(FILE *stream);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations.

A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *getw*( ) when access to the function is serialized.

getwc()

Get a wide character from a stream.

```
#include <stdio.h>
#include <wchar.h>

wint_t getwc(FILE *stream);
```

Issue 5: The Optional Header (OH) marking is removed from **<stdio.h>**.

getwchar()

Get a wide character from a *stdin* stream.

```
#include <wchar.h>

wint_t getwchar(void);
```

Issue 5: No functional changes in this issue.

getwd()

Get the current working directory pathname.

```
#include <unistd.h>

char *getwd(char *path_name);
```

Issue 5: No functional changes in this issue.

glob, globfree( )

Generate pathnames matching a pattern.

```
#include <glob.h>

int glob(const char *pattern, int flags,
    int(*errfunc)(const char *epath, int errno), glob_t *pglob);
void globfree(glob_t *pglob);
```

Issue 5: No functional changes in this issue.

gmtime( ), gmtime_r( )

Convert a time value to a broken-down UTC time.

```
#include <time.h>

struct tm *gmtime(const time_t *timer);
struct tm *gmtime_r(const time_t *clock, struct tm *result);
```

Issue 5: The *gmtime_r*( ) function is included for alignment with the POSIX Threads Extension, and is the thread-safe equivalent of *gmtime*( ).

A note indicating that the *gmtime*( ) interface need not be reentrant is added to the Description. A portable multi-threaded application should call *gmtime_r*( ) instead of *gmtime*( ).

grantpt( )

Grant access to the slave pseudo-terminal device.

```
#include <stdlib.h>

int grantpt(int fildes);
```

Issue 5: The last paragraph of the Description is moved from the Application Usage section. This notes that the behavior of the *grantpt*( ) function is unspecified if the application has installed a signal handler to catch SIGCHLD signals.

hcreate( ), hdestroy( ), hsearch( )

Manage hash search table.

```
#include <search.h>

int hcreate(size_t nel);
void hdestroy(void);
ENTRY *hsearch (ENTRY item, ACTION action);
```

Issue 5: No functional changes in this issue.

hypot( )

Euclidean distance function.

```
#include <math.h>

double hypot(double x, double y);
```

Issue 5: No functional changes in this issue.

iconv( )

Codeset conversion function.

```
#include <iconv.h>

size_t iconv(iconv_t cd, const char **inbuf, size_t *inbytesleft,
    char **outbuf, size_t *outbytesleft);
```

Issue 5: No functional changes in this issue.

iconv_close( )

Codeset conversion deallocation function.

```
#include <iconv.h>

int iconv_close(iconv_t cd);
```

Issue 5: No functional changes in this issue.

iconv_open( )

Codeset conversion allocation function.

```
#include <iconv.h>

iconv_t iconv_open(const char *tocode, const char *fromcode);
```

Issue 5: No functional changes in this issue.

ilogb( )

Return an unbiased exponent.

```
#include <math.h>

int ilogb (double x)
```

Issue 5: No functional changes in this issue.

index( )

Character string operations.

```
#include <strings.h>

char *index(const char *s, int c);
```

Issue 5: No functional changes in this issue.

initstate( ), random( ), setstate( ), srandom( )

Pseudo-random number functions.

```
#include <stdlib.h>

char *initstate(unsigned int seed, char *state, size_t size);
long random(void);
char *setstate(const char *state);
void srandom(unsigned int seed);
```

Issue 5: For *initstate*( ) in the Description section, the phrase ‘‘values smaller than 8’’ is replaced with ‘‘values greater than or equal to 8, or less than 32’’; ‘‘size<8’’ is

replaced with ‘‘size>=8 and <32’’; and a new first paragraph is added to the Return Value section. A note is added to the Application Usage section indicating that these changes restore the historical behavior of the function.

insque( ), remque( )

Insert or remove an element in a queue.

```
#include <search.h>

void insque(void *element, void *pred);
void remque(void *element);
```

Issue 5: No functional changes in this issue.

ioctl( )

Control a STREAMS device.

```
#include <stropts.h>

int ioctl(int fildes, int request, ... /* arg */);
```

Issue 5: No functional changes in this issue.

isalnum( )

Test for an alphanumeric character.

```
#include <ctype.h>

int isalnum(int c);
```

Issue 5: No functional changes in this issue.

isalpha( )

Test for an alphabetic character.

```
#include <ctype.h>

int isalpha(int c);
```

Issue 5: No functional changes in this issue.

isascii( )

Test for a 7-bit US-ASCII character.

```
#include <ctype.h>

int isascii(int c);
```

Issue 5: No functional changes in this issue.

isastream( )

Test a file descriptor.

```
#include <stropts.h>
int isastream(int fildes);
```

Issue 5: No functional changes in this issue.

isatty( )

Test for a terminal device.

```
#include <unistd.h>
int isatty(int fildes);
```

Issue 5: No functional changes in this issue.

iscntrl( )

Test for a control character.

```
#include <ctype.h>
int iscntrl(int c);
```

Issue 5: No functional changes in this issue.

isdigit( )

Test for a decimal digit.

```
#include <ctype.h>
int isdigit(int c);
```

Issue 5: No functional changes in this issue.

isgraph( )

Test for a visible character.

```
#include <ctype.h>
int isgraph(int c);
```

Issue 5: No functional changes in this issue.

islower( )

Test for a lowercase letter.

```
#include <ctype.h>
int islower(int c);
```

Issue 5: No functional changes in this issue.

isnan( )

Test for a NaN.

```
#include <math.h>
int isnan(double x);
```

Issue 5: No functional changes in this issue.

isprint( )

Test for a printing character.

```
#include <ctype.h>
int isprint(int c);
```

Issue 5: No functional changes in this issue.

ispunct( )

Test for a punctuation character.

```
#include <ctype.h>
int ispunct(int c);
```

Issue 5: No functional changes in this issue.

isspace( )

Test for a white-space character.

```
#include <ctype.h>
int isspace(int c);
```

Issue 5: No functional changes in this issue.

isupper( )

Test for an uppercase letter.

```
#include <ctype.h>
int isupper(int c);
```

Issue 5: No functional changes in this issue.

iswalnum( )

Test for an alphanumeric wide-character code.

```
#include <wctype.h>
int iswalnum(wint_t wc);
```

Issue 5: The following change has been made in this issue for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

- The Synopsis has been changed to indicate that this function and associated data types are now made visible by inclusion of the header **<wctype.h>** rather than **<wchar.h>**.

iswalpha( )

Test for an alphabetic wide-character code.

```
#include <wctype.h>

int iswalpha(wint_t wc);
```

Issue 5: The following change has been made in this issue for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

- The Synopsis has been changed to indicate that this function and associated data types are now made visible by inclusion of the header **<wctype.h>** rather than **<wchar.h>**.

iswcntrl( )

Test for a control wide-character code.

```
#include <wctype.h>

int iswcntrl(wint_t wc);
```

Issue 5: The following change has been made in this issue for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

- The Synopsis has been changed to indicate that this function and associated data types are now made visible by inclusion of the header **<wctype.h>** rather than **<wchar.h>**.

iswctype( )

Test character for a specified class.

```
#include <wctype.h>

int iswctype(wint_t wc, wctype_t charclass);
```

Issue 5: The following change has been made in this issue for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

- The Synopsis has been changed to indicate that this function and associated data types are now made visible by inclusion of the header **<wctype.h>** rather than **<wchar.h>**.

iswdigit( )

Test for a decimal digit wide-character code.

```
#include <wctype.h>

int iswdigit(wint_t wc);
```

Issue 5: The following change has been made in this issue for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

- The Synopsis has been changed to indicate that this function and associated data types are now made visible by inclusion of the header **<wctype.h>** rather than **<wchar.h>**.

iswgraph( )

Test for a visible wide-character code.

```
#include <wctype.h>

int iswgraph(wint_t wc);
```

Issue 5: The following change has been made in this issue for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

- The Synopsis has been changed to indicate that this function and associated data types are now made visible by inclusion of the header **<wctype.h>** rather than **<wchar.h>**.

iswlower( )

Test for a lowercase letter wide-character code.

```
#include <wctype.h>

int iswlower(wint_t wc);
```

Issue 5: The following change has been made in this issue for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

- The Synopsis has been changed to indicate that this function and associated data types are now made visible by inclusion of the header **<wctype.h>** rather than **<wchar.h>**.

iswprint( )

Test for a printing wide-character code.

```
#include <wctype.h>

int iswprint(wint_t wc);
```

Issue 5: The following change has been made in this issue for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

- The Synopsis has been changed to indicate that this function and associated data types are now made visible by inclusion of the header **<wctype.h>** rather than **<wchar.h>**.

iswpunct( )

Test for a punctuation wide-character code.

```
#include <wctype.h>

int iswpunct(wint_t wc);
```

Issue 5: The following change has been made in this issue for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

- The Synopsis has been changed to indicate that this function and associated data types are now made visible by inclusion of the header **<wctype.h>** rather than **<wchar.h>**.

iswspace( )

Test for a white-space wide-character code.

```
#include <wctype.h>

int iswspace(wint_t wc);
```

Issue 5: The following change has been made in this issue for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

- The Synopsis has been changed to indicate that this function and associated data types are now made visible by inclusion of the header **<wctype.h>** rather than **<wchar.h>**.

iswupper( )

Test for an uppercase letter wide-character code.

```
#include <wctype.h>

int iswupper(wint_t wc);
```

Issue 5: The following change has been made in this issue for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

- The Synopsis has been changed to indicate that this function and associated data types are now made visible by inclusion of the header **<wctype.h>** rather than **<wchar.h>**.

iswxdigit( )

Test for a hexadecimal digit wide-character code.

```
#include <wctype.h>

int iswxdigit(wint_t wc);
```

Issue 5: The following change has been made in this issue for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

- The Synopsis has been changed to indicate that this function and associated data types are now made visible by inclusion of the header **<wctype.h>** rather than **<wchar.h>**.

isxdigit( )

Test for a hexadecimal digit.

```
#include <ctype.h>

int isxdigit(int c);
```

Issue 5: No functional changes in this issue.

j0(), j1(), jn()

Bessel functions of the first kind.

```
#include <math.h>

double j0(double x);
double j1(double x);
double jn(int n, double x);
```

Issue 5: No functional changes in this issue.

jrand48()

Generate a uniformly distributed pseudo-random long signed integer.

```
#include <stdlib.h>

long int jrand48(unsigned short int xsubi[3]);
```

Issue 5: No functional changes in this issue.

kill()

Send a signal to a process or a group of processes.

```
#include <sys/types.h>
#include <signal.h>

int kill(pid_t pid, int sig);
```

Issue 5: The Description is updated for alignment with the POSIX Threads Extension. Specifically, the following paragraph is added:

''If the value of *pid* causes *sig* to be generated for the sending process, and if *sig* is not blocked for the calling thread and if no other thread has *sig* unblocked or is waiting in a *sigwait*( ) function for *sig*, either *sig* or at least one pending unblocked signal will be delivered to the sending thread before *kill*( ) returns.''

killpg()

Send a signal to a process group.

```
#include <signal.h>

int killpg(pid_t pgrp, int sig);
```

Issue 5: No functional changes in this issue.

l64a()

Convert a 32-bit integer to a radix-64 ASCII string.

```
#include <stdlib.h>

char *l64a(long value);
```

Issue 5: No functional changes in this issue.

labs( )

Return a long integer absolute value.

```
#include <stdlib.h>
long int labs(long int i);
```

Issue 5: No functional changes in this issue.

lchown( )

Change the owner and group of a symbolic link.

```
#include <unistd.h>
int lchown(const char *path, uid_t owner, gid_t group);
```

Issue 5: No functional changes in this issue.

lcong48( )

Seed a uniformly distributed pseudo-random signed long integer generator.

```
#include <stdlib.h>
void lcong48(unsigned short int param[7]);
```

Issue 5: No functional changes in this issue.

ldexp( )

Load exponent of a floating point number.

```
#include <math.h>
double ldexp(double x, int exp);
```

Issue 5: No functional changes in this issue.

ldiv( )

Compute quotient and remainder of a long division.

```
#include <stdlib.h>
ldiv_t ldiv(long int numer, long int denom);
```

Issue 5: No functional changes in this issue.

lfind( )

Find entry in a linear search table.

```
#include <search.h>
void *lfind(const void *key, const void *base, size_t *nelp,
    size_t width, int (*compar)(const void *, const void *));
```

Issue 5: No functional changes in this issue.

lgamma( )

Log gamma function.

```
#include <math.h>

double lgamma(double x);
extern int signgam;
```

Issue 5: A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *lgamma*( ) when access to the function is serialized.

link( )

Link to a file.

```
#include <unistd.h>

int link(const char *path1, const char *path2);
```

Issue 5: No functional changes in this issue.

lio_listio( )

List directed I/O. (**REALTIME**)

```
#include <aio.h>

int lio_listio(int mode, struct aiocb * const list[], int nent,
    struct sigevent *sig);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *lio_listio*( ) function allows the calling process to initiate a list of I/O requests with a single function call.

The *mode* argument takes one of the values LIO_WAIT or LIO_NOWAIT declared in **<aio.h>**, and determines whether the function returns when the I/O operations have been completed, or as soon as the operations have been queued.

Large File Support extensions are added; see Chapter 17 on page 163.

loc1( ), loc2( )

Pointers to characters matched by regular expressions. (**LEGACY**)

```
#include <regexp.h>

extern char *loc1;
extern char *loc2;
```

Issue 5: This symbols are marked LEGACY and may not be available on all implementations.

localeconv( )

Determine the program locale.

```
#include <locale.h>

struct lconv *localeconv(void);
```

Issue 5: No functional changes in this issue.

localtime( ), localtime_r( )

Convert a time value to a broken-down local time.

```
#include <time.h>

struct tm *localtime(const time_t *timer);
struct tm *localtime_r(const time_t *clock, struct tm *result);
```

Issue 5: The *localtime_r*( ) function is included for alignment with the POSIX Threads Extension, and is the thread-safe equivalent of *localtime*( ).

A note indicating that the *localtime*( ) interface need not be reentrant is added to the Description. A portable multi-threaded application should call *localtime_r*( ) instead of *localtime*( ).

lockf( )

Record locking on files.

```
#include <unistd.h>

int lockf(int fildes, int function, off_t size);
```

Issue 5: Large File Support extensions are added; see Chapter 17 on page 163. Specifically, the description of [EINVAL] is clarified and moved from optional to mandatory status.

A note is added to the Description indicating the effects of attempting to lock a section of a file that is associated with a buffered stream.

locs( )

Stop regular expression matching in a string. (**LEGACY**)

```
#include <regexp.h>

extern char *locs;
```

Issue 5: This symbol is marked LEGACY and may not be available on all implementations.

log( )

Natural logarithm function.

```
#include <math.h>

double log(double x);
```

Issue 5: No functional changes in this issue.

log10( )

Base 10 logarithm function.

```
#include <math.h>

double log10(double x);
```

Issue 5: No functional changes in this issue.

log1p( )

Compute a natural logarithm.

```
#include <math.h>

double log1p (double x);
```

Issue 5: No functional changes in this issue.

logb( )

Radix-independent exponent.

```
#include <math.h>

double logb(double x);
```

Issue 5: No functional changes in this issue.

_longjmp( ), _setjmp( )

Non-local goto.

```
#include <setjmp.h>

void _longjmp(jmp_buf env, int val);
int _setjmp(jmp_buf env);
```

Issue 5: No functional changes in this issue.

longjmp( )

Non-local goto.

```
#include <setjmp.h>

void longjmp(jmp_buf env, int val);
```

Issue 5: The Description is updated for alignment with the POSIX Threads Extension. Specifically, the effect of a call to *longjmp*( ) where initialization of the **jmp_buf** structure was not performed in the calling thread is undefined.

lrand48( )

Generate uniformly distributed pseudo-random non-negative long integers.

```
#include <stdlib.h>

long int lrand48(void);
```

Issue 5: No functional changes in this issue.

lsearch(), lfind()

Linear search and update.

```
#include <search.h>

void *lsearch(const void *key, void *base, size_t *nelp, size_t width,
    int (*compar)(const void *, const void *));
void *lfind(const void *key, const void *base, size_t *nelp,
    size_t width, int (*compar)(const void *, const void *));
```

Issue 5: No functional changes in this issue.

lseek()

Move the read-write file offset.

```
#include <unistd.h>

off_t lseek(int fildes, off_t offset, int whence);
```

Issue 5: The Description is updated for alignment with the POSIX Realtime Extension. Specifically, if *fildes* refers to a shared memory object, the behavior is unspecified.

Large File Support extensions are added; see Chapter 17 on page 163.

lstat()

Get symbolic link status.

```
#include <sys/stat.h>

int lstat(const char *path, struct stat *buf);
```

Issue 5: Large File Support extensions are added; see Chapter 17 on page 163.

makecontext(), swapcontext()

Manipulate user contexts.

```
#include <ucontext.h>

void makecontext(ucontext_t *ucp, (void *func)(), int argc, ...);
int swapcontext(ucontext_t *oucp, const ucontext_t *ucp);
```

Issue 5: In the Errors section, the description of [ENOMEM] is changed to apply to *swapcontext*( ) only.

malloc()

A memory allocator.

```
#include <stdlib.h>

void *malloc(size_t size);
```

Issue 5: No functional changes in this issue.

mblen( )

Get number of bytes in a character.

```
#include <stdlib.h>

int mblen(const char *s, size_t n);
```

Issue 5: No functional changes in this issue.

mbrlen( )

Get number of bytes in a character (restartable).

```
#include <wchar.h>

size_t mbrlen(const char *s, size_t n, mbstate_t *ps);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The *mbrlen*( ) function determines the number of bytes constituting the character pointed to by *s*. It is equivalent to:

```
mbstate_t internal;
mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);
```

mbrtowc( )

Convert a character to a wide-character code (restartable).

```
#include <wchar.h>

size_t mbrtowc(wchar_t *pwc, const char *s, size_t n, mbstate_t *ps);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The *mbrtowc*( ) function is equivalent to the call:

```
mbrtowc(NULL, ‘‘’’, 1, ps)
```

mbsinit( )

Determine conversion object status.

```
#include <wchar.h>

int mbsinit(const mbstate_t *ps);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The *mbsinit*( ) function determines whether the object pointed to by *ps* describes an initial conversion state.

mbsrtowcs( )

Convert a character string to a wide-character string (restartable).

```
#include <wchar.h>

size_t mbsrtowcs(wchar_t *dst, const char **src, size_t len,
    mbstate_t *ps);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The *mbsrtowcs*( ) function converts a sequence of characters, beginning in the conversion state described by the object pointed to by *ps*, from the array indirectly pointed to by *src* into a sequence of corresponding wide-characters.

mbstowcs( )

Convert a character string to a wide-character string.

```
#include <stdlib.h>

size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);
```

Issue 5: No functional changes in this issue.

mbtowc( )

Convert a character to a wide-character code.

```
#include <stdlib.h>

int mbtowc(wchar_t *pwc, const char *s, size_t n);
```

Issue 5: No functional changes in this issue.

memccpy( )

Copy bytes in memory.

```
#include <string.h>

void *memccpy(void *s1, const void *s2, int c, size_t n);
```

Issue 5: No functional changes in this issue.

memchr( )

Find byte in memory.

```
#include <string.h>

void *memchr(const void *s, int c, size_t n);
```

Issue 5: No functional changes in this issue.

memcmp( )

Compare bytes in memory.

```
#include <string.h>

int memcmp(const void *s1, const void *s2, size_t n);
```

Issue 5: No functional changes in this issue.

memcpy( )

Copy bytes in memory.

```
#include <string.h>

void *memcpy(void *s1, const void *s2, size_t n);
```

Issue 5: No functional changes in this issue.

memmove( )

Copy bytes in memory with overlapping areas.

```
#include <string.h>

void *memmove(void *s1, const void *s2, size_t n);
```

Issue 5: No functional changes in this issue.

memset( )

Set bytes in memory.

```
#include <string.h>

void *memset(void *s, int c, size_t n);
```

Issue 5: No functional changes in this issue.

mkdir( )

Make a directory.

```
#include <sys/types.h>
#include <sys/stat.h>

int mkdir(const char *path, mode_t mode);
```

Issue 5: No functional changes in this issue.

mkfifo( )

Make a FIFO special file.

```
#include <sys/types.h>
#include <sys/stat.h>

int mkfifo(const char *path, mode_t mode);
```

Issue 5: No functional changes in this issue.

mknod( )

Make a directory, a special, or regular file.

```
#include <sys/stat.h>

int mknod(const char *path, mode_t mode, dev_t dev);
```

Issue 5: No functional changes in this issue.

mkstemp( )

Make a unique filename.

```
#include <stdlib.h>

int mkstemp(char *template);
```

Issue 5: No functional changes in this issue.

mktemp( )

Make a unique filename.

```
#include <stdlib.h>

char *mktemp(char *template);
```

Issue 5: No functional changes in this issue.

mktime( )

Convert broken-down time into time since the Epoch.

```
#include <time.h>

time_t mktime(struct tm *timeptr);
```

Issue 5: No functional changes in this issue.

mlock( ), munlock( )

Lock or unlock a range of process address space. (**REALTIME**)

```
#include <sys/mman.h>

int mlock(const void * addr, size_t len);
int munlock(const void * addr, size_t len);
```

Issue 5: These are new functions included for alignment with the POSIX Realtime Extension. These are part of the Realtime Feature Group and may not be available on all implementations.

The function *mlock*( ) causes those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes to be memory resident until unlocked or until the process exits or *exec*( )s another process image. The implementation may require that *addr* be a multiple of PAGESIZE.

The function *munlock*( ) unlocks those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes, regardless of how many times *mlock*( ) has been called by the process for any of the pages in the specified range. The implementation may require that *addr*

be a multiple of the {PAGESIZE}.

mlockall( ), munlockall( )

Lock/unlock the address space of a process. (**REALTIME**)

```
#include <sys/mman.h>

int mlockall(int flags);
int munlockall(void);
```

Issue 5: These are new functions included for alignment with the POSIX Realtime Extension. These are part of the Realtime Feature Group and may not be available on all implementations.

The function *mlockall*( ) causes all of the pages mapped by the address space of a process to be memory resident until unlocked or until the process exits or *exec*( )s another process image.

The *flags* argument is constructed from the inclusive OR of one or more of the following symbolic constants, defined in **<sys/mman.h>**:

MCL_CURRENT
Lock all of the pages currently mapped into the address space of the process.

MCL_FUTURE
Lock all of the pages that become mapped into the address space of the process in the future, when those mappings are established.

The *munlockall*( ) function unlocks all currently mapped pages of the address space of the process. Any pages that become mapped into the address space of the process after a call to *munlockall*( ) will not be locked, unless there is an intervening call to *mlockall*( ) specifying MCL_FUTURE or a subsequent call to *mlockall*( ) specifying MCL_CURRENT.

mmap( )

Map pages of memory.

```
#include <sys/mman.h>

void *mmap(void *addr, size_t len, int prot, int flags,
    int fildes, off_t off);
```

Issue 5: Aligned with *mmap*( ) in the POSIX Realtime Extension. Specifically, the Description is extensively reworded; [EAGAIN] and [ENOTSUP] are added to the mandatory errors; and new cases of [ENOMEM] and [ENXIO] are added to the mandatory errors. Also, the value returned on failure is the value of the constant MAP_FAILED; this was previously defined as −1.

Large File Support extensions are added; see Chapter 17 on page 163.

modf( )

Decompose a floating-point number.

```
#include <math.h>

double modf(double x, double *iptr);
```

Issue 5: No functional changes in this issue.

mprotect( )

Set protection of memory mapping.

```
#include <sys/mman.h>

int mprotect(void *addr, size_t len, int prot);
```

Issue 5: Aligned with *mprotect*( ) in the POSIX Realtime Extension. Specifically, the Description is largely reworded; [ENOTSUP] and a second form of [ENOMEM] are added to the mandatory errors; and [EAGAIN] is moved from the optional to the mandatory errors.

mq_close( )

Close a message queue. (**REALTIME**)

```
#include <mqueue.h>

int mq_close(mqd_t mqdes);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *mq_close*( ) function removes the association between the message queue descriptor, *mqdes*, and its message queue.

mq_getattr( )

Get message queue attributes. (**REALTIME**)

```
#include <mqueue.h>

int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *mq_getattr*( ) function is used to get status information and attributes of the message queue and the open message queue description associated with the message queue descriptor.

mq_notify( )

Notify process that a message is available. (**REALTIME**)

```
#include <mqueue.h>

int mq_notify(mqd_t mqdes, const struct sigevent *notification);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

If the argument *notification* is not NULL, this function registers the calling process to be notified of message arrival at an empty message queue associated with the specified message queue descriptor, *mqdes*.

mq_open( )

Open a message queue. (**REALTIME**)

```
#include <mqueue.h>

mqd_t mq_open(const char *name, int oflag, ...);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *mq_open*( ) function establishes the connection between a process and a message queue with a message queue descriptor.

mq_receive( )

Receive a message from a message queue. (**REALTIME**)

```
#include <mqueue.h>

ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
    unsigned int *msg_prio);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *mq_receive*( ) function is used to receive the oldest of the highest priority message(s) from the message queue specified by *mqdes*.

mq_send( )

Send a message to a message queue. (**REALTIME**)

```
#include <mqueue.h>

int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
    unsigned int msg_prio);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *mq_send*( ) function adds the message pointed to by the argument *msg_ptr* to the message queue specified by *mqdes.*

mq_setattr( )

Set message queue attributes. (**REALTIME**)

```
#include <mqueue.h>

int mq_setattr(mqd_t mqdes, const struct mq_attr *mqstat,
    struct mq_attr *omqstat);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *mq_setattr*( ) function is used to set attributes associated with the open message queue description referenced by the message queue descriptor specified by *mqdes*.

mq_unlink( )

Remove a message queue. (**REALTIME**)

```
#include <mqueue.h>

int mq_unlink(const char *name);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *mq_unlink*( ) function removes the message queue named by the pathname *name*.

mrand48( )

Generate uniformly distributed pseudo-random signed long integers.

```
#include <stdlib.h>

long int mrand48(void);
```

Issue 5: No functional changes in this issue.

msgctl( )

Message control operations.

```
#include <sys/msg.h>

int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

Issue 5: The note about use of POSIX Realtime Extension IPC routines has been moved from Future Directions to a new Application Usage section, and reworded as follows:

> ‘‘The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines can be easily modified to use the alternative interfaces.’’

msgget( )

Get the message queue identifier.

```
#include <sys/msg.h>

int msgget(key_t key, int msgflg);
```

Issue 5: The note about use of POSIX Realtime Extension IPC routines has been moved from Future Directions to a new Application Usage section, and been reworded as follows:

> ‘‘The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines can be easily modified to use the alternative interfaces.’’

msgrcv( )

Message receive operation.

```
#include <sys/msg.h>

ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long int msgtyp,
    int msgflg);
```

Issue 5: The type of the return value is changed from **int** to **ssize_t**, and a warning is added to the Description about values of *msgsz* larger the {SSIZE_MAX}.

The note about use of POSIX Realtime Extension IPC routines has been moved from Future Directions to the Application Usage section, and reworded as follows:

> ‘‘The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines can be easily modified to use the alternative interfaces.’’

msgsnd( )

Message send operation.

```
#include <sys/msg.h>

int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

Issue 5: The note about use of POSIX Realtime Extension IPC routines has been moved from Future Directions to a new Application Usage section, and reworded as follows:

> ‘‘The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines can be easily modified to use the alternative interfaces.’’

msync( )

Synchronize memory with physical storage.

```
#include <sys/mman.h>

int msync(void *addr, size_t len, int flags);
```

Issue 5: Aligned with *msync*( ) in the POSIX Realtime Extension. Specifically, the Description is extensively reworded and [EBUSY] and a new form of [EINVAL] are added to the mandatory errors.

munlock( )

Unlock a range of process address space.

```
#include <sys/mman.h>

int munlock(const void * addr, size_t len);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

See *mlock*( ).

munlockall( )

Unlock the address space of a process.

```
#include <sys/mman.h>

int munlockall(void);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

See *mlockall*( ).

munmap( )

Unmap pages of memory.

```
#include <sys/mman.h>

int munmap(void *addr, size_t len);
```

Issue 5: Aligned with *munmap*( ) in the POSIX Realtime Extension. Specifically, the Description is extensively reworded and the SIGBUS error is no longer permitted to be generated.

nanosleep( )

High resolution sleep. (**REALTIME**)

```
#include <time.h>

int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *nanosleep*( ) function causes the current thread to be suspended from execution until either the time interval specified by the *rqtp* argument has elapsed or a signal is delivered to the calling thread and its action is to invoke a signal-catching function or to terminate the process.

nextafter( )

Next representable double-precision floating-point number.

```
#include <math.h>

double nextafter(double x, double y);
```

Issue 5: No functional changes in this issue.

nftw( )

Walk a file tree.

```
#include <ftw.h>

int nftw(const char *path, int (*fn)(const char *,
    const struct stat *, int, struct FTW *), int depth, int flags);
```

Issue 5: In the Description, the definition of the *depth* argument is clarified:

The argument *depth* sets the maximum number of file descriptors that will be used by *nftw*( ) while traversing the file tree. At most one file descriptor will be used for each directory level.

nice( )

Change nice value of a process.

```
#include <unistd.h>

int nice(int incr);
```

Issue 5: A statement is added to the description indicating the effects of this function on the different scheduling policies and multi-threaded processes.

The effect on processes or threads with other scheduling policies is implementation-dependent.

The nice value set with *nice*( ) is applied to the process. If the process is multi-threaded, the nice value affects all system scope threads in the process.

nl_langinfo( )

Language information.

```
#include <langinfo.h>

char *nl_langinfo(nl_item item);
```

Issue 5: The last paragraph of the Description, describing that the return value may overwrite an array, is moved from the Application Usage section.

A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *nl_langinfo*( ) when access to the function is serialized.

nrand48( )

Generate uniformly distributed pseudo-random non-negative long integers.

```
#include <stdlib.h>

long int nrand48(unsigned short int xsubi[3]);
```

Issue 5: No functional changes in this issue.

open( )

Open a file.

```
#include <fcntl.h>

int open(const char *path, int oflag, ...);
```

Issue 5: The Description is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension. Specifically, the O_DSYNC and O_RSYNC flags are added and the semantics of the *open*( ) function on the master side of a pseudo-terminal device are better described.

Large File Support extensions are added; see Chapter 17 on page 163.

opendir( )

Open a directory.

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char *dirname);
```

Issue 5: No functional changes in this issue.

openlog( )

Open a connection to the logging facility.

```
#include <syslog.h>

void openlog(const char *ident, int logopt, int facility);
```

Issue 5: No functional changes in this issue.

optarg( ), opterr( ), optind( ), optopt( )

Options parsing variables.

```
#include <stdio.h>

extern char *optarg;
extern int opterr, optind, optopt;
```

Issue 5: No functional changes in this issue.

pathconf( )

Get configurable pathname variables.

```
#include <unistd.h>

long int pathconf(const char *path, int name);
```

Issue 5: The Description is updated for alignment with the POSIX Realtime Extension. See *fpathconf*( ).

Large File Support extensions are added; see Chapter 17 on page 163.

pause( )

Suspend the thread until a signal is received.

```
#include <unistd.h>

int pause(void);
```

Issue 5: The Description is updated for alignment with the POSIX Threads Extension. Specifically, this is now described in terms of the thread rather than the process.

pclose( )

Close a pipe stream to or from a process.

```
#include <stdio.h>

int pclose(FILE *stream);
```

Issue 5: No functional changes in this issue.

perror( )

Write error messages to standard error.

```
#include <stdio.h>

void perror(const char *s);
```

Issue 5: A paragraph is added to the Description indicating that *perror*( ) does not change the orientation of the standard error stream.

pipe( )

Create an interprocess channel.

```
#include <unistd.h>

int pipe(int fildes[2]);
```

Issue 5: No functional changes in this issue.

poll( )

Input/output multiplexing.

```
#include <poll.h>

int poll(struct pollfd fds[], nfds_t nfds, int timeout);
```

Issue 5: The description of POLLWRBAND is updated.

popen( )

Initiate pipe streams to or from a process.

```
#include <stdio.h>

FILE *popen(const char *command, const char *mode);
```

Issue 5: A statement is added to the Description indicating that pipe streams are byte-oriented.

pow( )

Power function.

```
#include <math.h>

double pow(double x, double y);
```

Issue 5: No functional changes in this issue.

pread( )

Read from a file.

```
#include <unistd.h>

ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);
```

Issue 5: New in Issue 5; see *read*( ).

printf( )

Print formatted output.

```
#include <stdio.h>

int printf(const char *format, ...);
```

Issue 5: No functional changes in this issue.

pthread_atfork( )

Register fork handlers.

```
#include <sys/types.h>
#include <unistd.h>

int pthread_atfork(void (*prepare)(void), void (*parent)(void),
    void (*child)(void));
```

Issue 5: This is a new function derived from the POSIX Threads Extension, including IEEE Std 1003.1c-1995 #4.

The *pthread_atfork*( ) function declares fork handlers to be called before and after *fork*( ), in the context of the thread that called *fork*( ).

pthread_attr_getguardsize( ), pthread_attr_setguardsize( )

Get or set the thread guardsize attribute.

```
#include <pthread.h>

int pthread_attr_getguardsize(const pthread_attr_t *attr,
    size_t *guardsize);
int pthread_attr_setguardsize(pthread_attr_t *attr,
    size_t guardsize);
```

Issue 5: These are included as part of the X/Open Threads Extension; see Chapter 12 on page 109.

The *pthread_attr_getguardsize*( ) function gets the *guardsize* attribute in the *attr* object. This attribute is returned in the *guardsize* parameter.

The *pthread_attr_setguardsize*( ) function sets the *guardsize* attribute in the *attr* object. The new value of this attribute is obtained from the *guardsize* parameter.

pthread_attr_init( ), pthread_attr_destroy( )

Initialize and destroy threads attribute object.

```
#include <pthread.h>

int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The function *pthread_attr_init*( ) initializes a thread attributes object *attr* with the default value for all of the individual attributes used by a given implementation.

The *pthread_attr_destroy*( ) function is used to destroy a thread attributes object.

pthread_attr_setdetachstate( ), pthread_attr_getdetachstate( )

Set and get the detachstate attribute.

```
#include <pthread.h>

int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
int pthread_attr_getdetachstate(const pthread_attr_t *attr,
    int *detachstate);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The functions *pthread_attr_setdetachstate*( ) and *pthread_attr_getdetachstate*( ), respectively, set and get the *detachstate* attribute in the *attr* object.

pthread_attr_setinheritsched( ), pthread_attr_getinheritsched( )

Set and get the inheritsched attribute. (**REALTIME THREADS**)

```
#include <pthread.h>

int pthread_attr_setinheritsched(pthread_attr_t *attr,
    int inheritsched);
int pthread_attr_getinheritsched(const pthread_attr_t *attr,
    int *inheritsched);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

These are included as part of the Realtime Threads Feature Group, and may not be available on all implementations.

The functions *pthread_attr_setinheritsched*( ) and *pthread_attr_getinheritsched*( ), respectively, set and get the *inheritsched* attribute in the *attr* argument.

pthread_attr_setschedparam( ), pthread_attr_getschedparam( )

Set and get the schedparam attribute. (**REALTIME THREADS**)

```
#include <pthread.h>

int pthread_attr_setschedparam(pthread_attr_t *attr,
    const struct sched_param *param);
int pthread_attr_getschedparam(const pthread_attr_t *attr,
    struct sched_param *param);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The functions *pthread_attr_setschedparam*( ) and *pthread_attr_getschedparam*( ), respectively, set and get the scheduling parameter attributes in the *attr* argument.

pthread_attr_setschedpolicy( ), pthread_attr_getschedpolicy( )

Set and get the schedpolicy attribute. (**REALTIME THREADS**)

```
#include <pthread.h>

int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
int pthread_attr_getschedpolicy(const pthread_attr_t *attr,
    int *policy);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

These are included as part of the Realtime Threads Feature Group, and may not be available on all implementations.

The functions *pthread_attr_setschedpolicy*( ) and *pthread_attr_getschedpolicy*( ), respectively, set and get the *schedpolicy* attribute in the *attr* argument.

pthread_attr_setscope( ), pthread_attr_getscope( )

Set and get the contentionscope attribute. (**REALTIME THREADS**)

```
#include <pthread.h>

int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
int pthread_attr_getscope(const pthread_attr_t *attr,
    int *contentionscope);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

These are included as part of the Realtime Threads Feature Group, and may not be available on all implementations.

The *pthread_attr_setscope*( ) and *pthread_attr_getscope*( ) functions are used to set and get the *contentionscope* attribute in the *attr* object.

pthread_attr_setstackaddr( ), pthread_attr_getstackaddr( )

Set and get the stackaddr attribute.

```
#include <pthread.h>

int pthread_attr_setstackaddr(pthread_attr_t *attr, void *stackaddr);
int pthread_attr_getstackaddr(const pthread_attr_t *attr,
    void **stackaddr);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The functions *pthread_attr_setstackaddr*( ) and *pthread_attr_getstackaddr*( ), respectively, set and get the thread creation *stackaddr* attribute in the *attr* object.

pthread_attr_setstacksize( ), pthread_attr_getstacksize( )

Set and get the stacksize attribute.

```
#include <pthread.h>

int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
int pthread_attr_getstacksize(const pthread_attr_t *attr,
    size_t *stacksize);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The functions *pthread_attr_setstacksize*( ) and *pthread_attr_getstacksize*( ), respectively, set and get the thread creation *stacksize* attribute in the *attr* object.

pthread_cancel( )

Cancel execution of a thread.

```
#include <pthread.h>

int pthread_cancel(pthread_t thread);
```

Issue 5: This is a new function included for alignment with the POSIX Threads Extension.

The *pthread_cancel*( ) function requests that *thread* be canceled.

pthread_cleanup_push(), pthread_cleanup_pop()

Establish cancelation handlers.

```
#include <pthread.h>

void pthread_cleanup_push(void (*routine)(void*), void *arg);
void pthread_cleanup_pop(int execute);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The *pthread_cleanup_push*( ) function pushes the specified cancelation cleanup handler routine onto the calling thread's cancelation cleanup stack.

The *pthread_cleanup_pop*( ) function removes the routine at the top of the calling thread's cancelation cleanup stack and optionally invokes it (if *execute* is non-zero).

pthread_cond_init(), pthread_cond_destroy()

Initialize and destroy condition variables.

```
#include <pthread.h>

int pthread_cond_init(pthread_cond_t *cond,
    const pthread_condattr_t *attr);
int pthread_cond_destroy(pthread_cond_t *cond);
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The function *pthread_cond_init*( ) initializes the condition variable referenced by *cond* with attributes referenced by *attr*.

The function *pthread_cond_destroy*( ) destroys the given condition variable specified by *cond*; the object becomes, in effect, uninitialized. An implementation may cause *pthread_cond_destroy*( ) to set the object referenced by *cond* to an invalid value.

pthread_cond_signal(), pthread_cond_broadcast()

Signal or broadcast a condition.

```
#include <pthread.h>

int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The *pthread_cond_signal*( ) function unblocks at least one of the threads that are blocked on the specified condition variable *cond* (if any threads are blocked on *cond*).

The *pthread_cond_broadcast*( ) function unblocks all threads currently blocked on the specified condition variable *cond*.

pthread_cond_wait( ), pthread_cond_timedwait( )

Wait on a condition.

```
#include <pthread.h>

int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
int pthread_cond_timedwait(pthread_cond_t *cond,
    pthread_mutex_t *mutex, const struct timespec *abstime);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The *pthread_cond_wait*( ) and *pthread_cond_timedwait*( ) functions are used to block on a condition variable.

pthread_condattr_getpshared( ), pthread_condattr_setpshared( )

Get and set the process-shared condition variable attributes.

```
#include <pthread.h>

int pthread_condattr_getpshared(const pthread_condattr_t *attr,
    int *pshared);
int pthread_condattr_setpshared(pthread_condattr_t *attr,
    int pshared);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The *pthread_condattr_getpshared*( ) function obtains the value of the *process-shared* attribute from the attributes object referenced by *attr*.

The *pthread_condattr_setpshared*( ) function is used to set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

pthread_condattr_init( ), pthread_condattr_destroy( )

Initialize and destroy the condition variable attributes object.

```
#include <pthread.h>

int pthread_condattr_init(pthread_condattr_t *attr);
int pthread_condattr_destroy(pthread_condattr_t *attr);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The function *pthread_condattr_init*( ) initializes a condition variable attributes object *attr* with the default value for all of the attributes defined by the implementation.

The *pthread_condattr_destroy*( ) function destroys a condition variable attributes object; the object becomes, in effect, uninitialized.

pthread_create( )

Thread creation.

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
    void *(*start_routine)(void*), void *arg);
```

Issue 5: This is a new function included for alignment with the POSIX Threads Extension.

The *pthread_create*( ) function is used to create a new thread, with attributes specified by *attr*, within a process.

pthread_detach( )

Detach a thread.

```
#include <pthread.h>

int pthread_detach(pthread_t thread);
```

Issue 5: This is a new function included for alignment with the POSIX Threads Extension.

The *pthread_detach*( ) function is used to indicate to the implementation that storage for the thread *thread* can be reclaimed when that thread terminates.

pthread_equal( )

Compare thread IDs.

```
#include <pthread.h>

int pthread_equal(pthread_t t1, pthread_t t2);
```

Issue 5: This is a new function included for alignment with the POSIX Threads Extension.

This function compares the thread IDs *t1* and *t2*.

pthread_exit( )

Thread termination.

```
#include <pthread.h>

void pthread_exit(void *value_ptr);
```

Issue 5: This is a new function included for alignment with the POSIX Threads Extension.

The *pthread_exit*( ) function terminates the calling thread and makes the value *value_ptr* available to any successful join with the terminating thread. Any cancelation cleanup handlers that have been pushed and not yet popped are popped in the reverse order that they were pushed and then executed.

pthread_getconcurrency(), pthread_setconcurrency()

Get or set level of concurrency.

```
#include <pthread.h>

int pthread_getconcurrency(void);
int pthread_setconcurrency(int new_level);
```

Issue 5: These are included as part of the X/Open Threads Extension; see Chapter 12 on page 109.

The *pthread_setconcurrency*( ) function allows an application to inform the threads implementation of its desired concurrency level, *new_level*.

The *pthread_getconcurrency*( ) function returns the value set by a previous call to the *pthread_setconcurrency*( ) function.

pthread_getschedparam(), pthread_setschedparam()

Dynamic thread scheduling parameters access. (**REALTIME THREADS**)

```
#include <pthread.h>

int pthread_getschedparam(pthread_t thread, int *policy,
    struct sched_param *param);
int pthread_setschedparam(pthread_t thread, int *policy,
    const struct sched_param *param);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

These are included as part of the Realtime Threads Feature Group, and may not be available on all implementations.

The *pthread_getschedparam*( ) and *pthread_setschedparam*( ) functions allow the scheduling policy and scheduling parameters of individual threads within a multi-threaded process to be retrieved and set.

pthread_join()

Wait for thread termination.

```
#include <pthread.h>

int pthread_join(pthread_t thread, void **value_ptr);
```

Issue 5: This is a new function included for alignment with the POSIX Threads Extension.

The *pthread_join*( ) function suspends execution of the calling thread until the target *thread* terminates, unless the target *thread* has already terminated.

pthread_key_create()

Thread-specific data key creation.

```
#include <pthread.h>

int pthread_key_create(pthread_key_t *key, void (*destructor)(void*));
```

Issue 5: This is a new function included for alignment with the POSIX Threads Extension.

This function creates a thread-specific data key visible to all threads in the process.

pthread_key_delete( )

Thread-specific data key deletion.

```
#include <pthread.h>

int pthread_key_delete(pthread_key_t key);
```

Issue 5: This is a new function included for alignment with the POSIX Threads Extension.

This function deletes a thread-specific data key previously returned by *pthread_key_create*( ).

pthread_kill( )

Send a signal to a thread.

```
#include <signal.h>

int pthread_kill(pthread_t thread, int sig);
```

Issue 5: This is a new function included for alignment with the POSIX Threads Extension.

The *pthread_kill*( ) function is used to request that a signal be delivered to the specified thread.

pthread_mutex_init( ), pthread_mutex_destroy( )

Initialize or destroy a mutex.

```
#include <pthread.h>

int pthread_mutex_init(pthread_mutex_t *mutex,
    const pthread_mutexattr_t *attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The *pthread_mutex_init*( ) function initializes the mutex referenced by *mutex* with attributes specified by *attr*.

The *pthread_mutex_destroy*( ) function destroys the mutex object referenced by *mutex*; the mutex object becomes, in effect, uninitialized. An implementation may cause *pthread_mutex_destroy*( ) to set the object referenced by *mutex* to an invalid value.

pthread_mutex_lock( ), pthread_mutex_trylock( ), pthread_mutex_unlock( )

Lock and unlock a mutex.

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The function *pthread_mutex_lock*( ) locks a mutex object referenced by *mutex*.

The function *pthread_mutex_trylock*( ) is identical to *pthread_mutex_lock*( ), except that if the mutex object referenced by *mutex* is currently locked (by any thread, including the current thread), the call returns immediately.

The *pthread_mutex_unlock*( ) function releases the mutex object referenced by *mutex*.

pthread_mutex_setprioceiling( ), pthread_mutex_getprioceiling( )

Change the priority ceiling of a mutex. (**REALTIME THREADS**)

```
#include <pthread.h>

int pthread_mutex_setprioceiling(pthread_mutex_t *mutex,
    int prioceiling, int *old_ceiling);
int pthread_mutex_getprioceiling(const pthread_mutex_t *mutex,
    int *prioceiling);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

These are included as part of the Realtime Threads Feature Group, and may not be available on all implementations.

The *pthread_mutex_getprioceiling*( ) function returns the current priority ceiling of the mutex.

The *pthread_mutex_setprioceiling*( ) function either locks the mutex if it is unlocked, or blocks until it can successfully lock the mutex, then it changes the mutex' priority ceiling and releases the mutex. When the change is successful, the previous value of the priority ceiling is returned in *old_ceiling*.

pthread_mutexattr_getpshared( ), pthread_mutexattr_setpshared( )

Set and get the process-shared attribute.

```
#include <pthread.h>

int pthread_mutexattr_getpshared(const pthread_mutexattr_t *attr,
    int *pshared);
int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
    int pshared);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The *pthread_mutexattr_getpshared*( ) function obtains the value of the *process-shared* attribute from the attributes object referenced by *attr*. The *pthread_mutexattr_setpshared*( ) function is used to set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

pthread_mutexattr_init(), pthread_mutexattr_destroy()

Initialize and destroy mutex attributes object.

```
#include <pthread.h>

int pthread_mutexattr_init(pthread_mutexattr_t *attr);
int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The function *pthread_mutexattr_init*( ) initializes a mutex attributes object *attr* with the default value for all of the attributes defined by the implementation.

The *pthread_mutexattr_destroy*( ) function destroys a mutex attributes object; the object becomes, in effect, uninitialized.

pthread_mutexattr_setprioceiling(), pthread_mutexattr_getprioceiling()

Set and get the prioceiling attribute of a mutex attribute object. (**REALTIME THREADS**)

```
#include <pthread.h>

int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
    int prioceiling);
int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t *attr,
    int *prioceiling);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

These are included as part of the Realtime Threads Feature Group, and may not be available on all implementations.

The *pthread_mutexattr_setprioceiling*( ) and *pthread_mutexattr_getprioceiling*( ) functions, respectively, set and get the priority ceiling attribute of a mutex attribute object pointed to by *attr* which was previously created by the function *pthread_mutexattr_init*( ).

pthread_mutexattr_setprotocol(), pthread_mutexattr_getprotocol()

Set and get the protocol attribute of a mutex attribute object. (**REALTIME THREADS**)

```
#include <pthread.h>

int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
    int protocol);
int pthread_mutexattr_getprotocol(const pthread_mutexattr_t *attr,
    int *protocol);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

These are included as part of the Realtime Threads Feature Group, and may not be available on all implementations.

The *pthread_mutexattr_setprotocol*( ) and *pthread_mutexattr_getprotocol*( ) functions, respectively, set and get the protocol attribute of a mutex attribute object pointed to by *attr* which was previously created by the function *pthread_mutexattr_init*( ).

pthread_mutexattr_gettype( ), pthread_mutexattr_settype( )

Get or set a mutex type.

```
#include <pthread.h>

int pthread_mutexattr_gettype(pthread_mutexattr_t *attr, int *type);
int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
```

Issue 5: These are included as part of the X/Open Threads Extension; see Chapter 12 on page 109.

The *pthread_mutexattr_gettype*( ) and *pthread_mutexattr_settype*( ) functions, respectively, get and set the mutex *type* attribute.

pthread_once( )

Dynamic package initialization.

```
#include <pthread.h>

int pthread_once(pthread_once_t *once_control,
    void (*init_routine)(void));
pthread_once_t once_control = PTHREAD_ONCE_INIT;
```

Issue 5: This is a new function included for alignment with the POSIX Threads Extension.

The first call to *pthread_once*( ) by any thread in a process, with a given *once_control*, will call the *init_routine*( ) with no arguments.

pthread_rwlock_init( ), pthread_rwlock_destroy( )

Initialize or destroy a read-write lock object.

```
#include <pthread.h>

int pthread_rwlock_init(pthread_rwlock_t *rwlock,
    const pthread_rwlockattr_t *attr);
int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
pthread_rwlock_t rwlock=PTHREAD_RWLOCK_INITIALIZER;
```

Issue 5: These are included as part of the X/Open Threads Extension; see Chapter 12 on page 109.

The *pthread_rwlock_init*( ) function initializes the read-write lock referenced by *rwlock* with the attributes referenced by *attr*.

The *pthread_rwlock_destroy*( ) function destroys the read-write lock object referenced by *rwlock* and releases any resources used by the lock.

pthread_rwlock_rdlock( ), pthread_rwlock_tryrdlock( )

Lock a read-write lock object for reading.

```
#include <pthread.h>

int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);
```

Issue 5: These are included as part of X/Open Threads Extension; see Chapter 12 on page 109.

The *pthread_rwlock_rdlock*( ) function applies a read lock to the read-write lock referenced by *rwlock*.

The function *pthread_rwlock_tryrdlock*( ) applies a read lock as in the *pthread_rwlock_rdlock*( ) function, with the exception that the function fails if any thread holds a write lock on *rwlock* or there are writers blocked on *rwlock*.

pthread_rwlock_unlock( )

Unlock a read-write lock object.

```
#include <pthread.h>

int pthread_rwlock_unlock(pthread_rwlock_t *rwlock);
```

Issue 5: This is included as part of X/Open Threads Extension; see Chapter 12 on page 109.

The *pthread_rwlock_unlock*( ) function is called to release a lock held on the read-write lock object referenced by *rwlock*.

pthread_rwlock_wrlock( ), pthread_rwlock_trywrlock( )

Lock a read-write lock object for writing.

```
#include <pthread.h>

int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlock);
```

Issue 5: These are included as part of the X/Open Threads Extension; see Chapter 12 on page 109.

The *pthread_rwlock_wrlock*( ) function applies a write lock to the read-write lock referenced by *rwlock*.

The function *pthread_rwlock_trywrlock*( ) applies a write lock like the *pthread_rwlock_wrlock*( ) function, with the exception that the function fails if any thread currently holds *rwlock* (for reading or writing).

pthread_rwlockattr_getpshared( ), pthread_rwlockattr_setpshared( )

Get and set the process-shared attribute of a read-write lock attributes object.

```
#include <pthread.h>

int pthread_rwlockattr_getpshared(const pthread_rwlockattr_t *attr,
    int *pshared);
int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
    int pshared);
```

Issue 5: These are included as part of the X/Open Threads Extension; see Chapter 12 on page 109.

The *pthread_rwlockattr_getpshared*( ) function obtains the value of the *process-shared* attribute from the initialized attributes object referenced by *attr*.

The *pthread_rwlockattr_setpshared*( ) function is used to set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

pthread_rwlockattr_init(), pthread_rwlockattr_destroy()

Initialize and destroy a read-write lock attributes object.

```
#include <pthread.h>

int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);
int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);
```

Issue 5: These are included as part of X/Open Threads Extension; see Chapter 12 on page 109.

The function *pthread_rwlockattr_init*( ) initializes a read-write lock attributes object *attr* with the default value for all of the attributes defined by the implementation.

The *pthread_rwlockattr_destroy*( ) function destroys a read-write lock attributes object. The effect of subsequent use of the object is undefined until the object is re-initialized by another call to *pthread_rwlockattr_init*( ).

pthread_self( )

Get calling thread's ID.

```
#include <pthread.h>

pthread_t pthread_self(void);
```

Issue 5: This is a new function included for alignment with the POSIX Threads Extension.

The *pthread_self*( ) function returns the thread ID of the calling thread.

pthread_setcancelstate( ), pthread_setcanceltype( ), pthread_testcancel( )

Set cancelability state.

```
#include <pthread.h>

int pthread_setcancelstate(int state, int *oldstate);
int pthread_setcanceltype(int type, int *oldtype);
void pthread_testcancel(void);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The *pthread_setcancelstate*( ) function atomically both sets the calling thread's cancelability state to the indicated *state* and returns the previous cancelability state at the location referenced by *oldstate*.

The *pthread_setcanceltype*( ) function atomically both sets the calling thread's cancelability type to the indicated *type* and returns the previous cancelability type at the location referenced by *oldtype*.

pthread_setconcurrency( )

Get or set the level of concurrency.

```
#include <pthread.h>

int pthread_setconcurrency(int new_level);
```

Issue 5: This is included as part of the X/Open Threads Extension; see Chapter 12 on page 109.

See *pthread_getconcurrency*( ).

pthread_setspecific( ), pthread_getspecific( )

Thread-specific data management.

```
#include <pthread.h>

int pthread_setspecific(pthread_key_t key, const void *value);
void *pthread_getspecific(pthread_key_t key);
```

Issue 5: These are new functions included for alignment with the POSIX Threads Extension.

The *pthread_setspecific*( ) function associates a thread-specific *value* with a *key* obtained via a previous call to *pthread_key_create*( ).

The *pthread_getspecific*( ) function returns the value currently bound to the specified *key* on behalf of the calling thread.

pthread_sigmask( )

Examine and change blocked signals.

```
#include <signal.h>

int pthread_sigmask(int how, const sigset_t *set, sigset_t *oset);
```

Issue 5: This is a new function included for alignment with the POSIX Threads Extension.

The *pthread_sigmask*( ) function is used to examine or change (or both) the calling thread's signal mask, regardless of the number of threads in the process.

ptsname( )

Get name of the slave pseudo-terminal device.

```
#include <stdlib.h>

char *ptsname(int fildes);
```

Issue 5: A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *ptsname*( ) when access to the function is serialized.

putc( )

Put byte on a stream.

```
#include <stdio.h>

int putc(int c, FILE *stream);
```

Issue 5: No functional changes in this issue.

putchar( )

Put byte on *stdout* stream.

```
#include <stdio.h>

int putchar(int c);
```

Issue 5: No functional changes in this issue.

putc_unlocked( )

Stdio with explicit client locking.

```
#include <stdio.h>

int putc_unlocked(int c, FILE *stream);
```

Issue 5: This is a new function included for alignment with the POSIX Threads Extension.

putchar_unlocked( )

Stdio with explicit client locking.

```
#include <stdio.h>

int putchar_unlocked(int c);
```

Issue 5: This is a new function included for alignment with the POSIX Threads Extension.

putenv( )

Change or add a value to the environment.

```
#include <stdlib.h>

int putenv(char *string);
```

Issue 5: The type of the argument to this function is changed from **const char*** to **char***. This was indicated as a Future Direction in previous issues.

A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *putenv*( ) when access to the function is serialized.

putmsg( ), putpmsg( )

Send a message on a STREAM.

```
#include <stropts.h>

int putmsg(int fildes, const struct strbuf *ctlptr,
    const struct strbuf *dataptr, int flags);
int putpmsg(int fildes, const struct strbuf *ctlptr,
    const struct strbuf *dataptr, int band, int flags);
```

Issue 5: The following line of text is removed from the Description: ‘‘The STREAM head guarantees that the control part of a message generated by *putmsg*( ) is at least 64 bytes in length.’’

puts( )

Put a string on standard output.

```
#include <stdio.h>

int puts(const char *s);
```

Issue 5: No functional changes in this issue.

pututxline( )

Put an entry into user accounting database.

```
#include <utmpx.h>

struct utmpx *pututxline(const struct utmpx *utmpx);
```

Issue 5: No functional changes in this issue.

putw( )

Put a word on a stream. (**LEGACY**)

```
#include <stdio.h>

int putw(int w, FILE *stream);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations.

A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *putw*( ) when access to the function is serialized.

putwc( )

Put a wide character on a stream.

```
#include <stdio.h>
#include <wchar.h>

wint_t putwc(wchar_t wc, FILE *stream);
```

Issue 5: Aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc* is changed from **wint_t** to **wchar_t**.

The Optional Header (OH) marking is removed from **<stdio.h>**.

putwchar( )

Put a wide character on a *stdout* stream.

```
#include <wchar.h>

wint_t putwchar(wchar_t wc);
```

Issue 5: Aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc* is changed from **wint_t** to **wchar_t**.

pwrite( )

Write on a file.

```
#include <unistd.h>

ssize_t pwrite(int fildes, const void *buf, size_t nbyte,
    off_t offset);
```

Issue 5: New in Issue 5; see *write*( ).

qsort( )

Sort a table of data.

```
#include <stdlib.h>

void qsort(void *base, size_t nel, size_t width
    int (*compar)(const void *, const void *));
```

Issue 5: No functional changes in this issue.

raise( )

Send a signal to the executing process.

```
#include <signal.h>

int raise(int sig);
```

Issue 5: The Description is updated for alignment with the POSIX Threads Extension. Specifically, it now describes the signal *sig* being sent to the executing thread.

The effect of the *raise*( ) function is equivalent to calling:

```
pthread_kill(pthread_self(), sig);
```

rand( ), rand_r( )

Pseudo-random number generator.

```
#include <stdlib.h>

int rand (void);
void srand(unsigned int seed);
int rand_r(unsigned int *seed);
```

Issue 5: The *rand_r*( ) function is included for alignment with the POSIX Threads Extension, and is the thread-safe equivalent of *rand*( ).

A note indicating that the *rand*( ) interface need not be reentrant is added to the Description. A portable multi-threaded application should call *rand_r*( ) instead of *rand*( ).

random( )

Generate a pseudo-random number.

```
#include <stdlib.h>

long random(void);
```

Issue 5: No functional changes in this issue.

read( ), readv( ), pread( )

Read from a file.

```
#include <unistd.h>

ssize_t read(int fildes, void *buf, size_t nbyte);
ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);

#include <sys/uio.h>

ssize_t readv(int fildes, const struct iovec *iov, int iovcnt);
```

Issue 5: The Description is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension. Specifically, parts of the description are worded in terms of a thread rather than a process. Also, semantics are described for synchronized input/output and shared memory objects.

Large File Support extensions are added; see Chapter 17 on page 163.

The *pread*( ) function is added.

The *pread*( ) function performs the same action as *read*( ), except that it reads from a given position in the file without changing the file pointer.

readdir( ), readdir_r( )

Read directory.

```
#include <sys/types.h>
#include <dirent.h>

struct dirent *readdir(DIR *dirp);
int readdir_r(DIR *dirp, struct direct *entry, struct dirent **result);
```

Issue 5: Large File Support extensions are added; see Chapter 17 on page 163.

The *readdir_r*( ) function is included for alignment with the POSIX Threads Extension, and is the thread-safe equivalent of *readdir*( ).

A note indicating that the *readdir*( ) interface need not be reentrant is added to the Description. A portable multi-threaded application should call *readdir_r*( ) instead of *readdir*( ).

readlink( )

Read the contents of a symbolic link.

```
#include <unistd.h>

int readlink(const char *path, char *buf, size_t bufsize);
```

Issue 5: The *readlink*( ) function returns the size of the information that it reads as a type **int**, but the size of the buffer area is specified by a type **size_t**. This function is being specified in the IEEE PASC P1003.1a draft standard, which currently has type **ssize_t** for the return type. A future issue will likely change for alignment.

readv( )

Vectored read from file.

```
#include <sys/uio.h>

ssize_t readv(int fildes, const struct iovec *iov, int iovcnt);
```

Issue 5: No functional changes in this issue.

realloc( )

Memory reallocator.

```
#include <stdlib.h>

void *realloc(void *ptr, size_t size);
```

Issue 5: No functional changes in this issue.

realpath( )

Resolve a pathname.

```
#include <stdlib.h>

char *realpath(const char *file_name, char *resolved_name);
```

Issue 5: No functional changes in this issue.

re_comp( ), re_exec( )

Compile and execute regular expressions. (**LEGACY**)

```
#include <re_comp.h>

char *re_comp(const char *string);
int re_exec(const char *string);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations.

A note indicating that these interfaces need not be reentrant is added to the Description. A portable multi-threaded application may only safely call these functions when access to the function is serialized.

regcmp(), regex()

Compile and execute a regular expression. (**LEGACY**)

```
#include <libgen.h>

char *regcmp (const char *string1, ... /*, (char *)0 */);
char *regex (const char *re, const char *subject, ...);
extern char *__loc1;
```

Issue 5: This function is marked LEGACY and may not be available on all implementations.

A note indicating that these interfaces need not be reentrant is added to the Description. A portable multi-threaded application may only safely call these functions when access to the function is serialized.

regcomp(), regexec(), regerror(), regfree()

Regular expression matching.

```
#include <sys/types.h>
#include <regex.h>

int regcomp(regex_t *preg, const char *pattern, int cflags);
int regexec(const regex_t *preg, const char *string,
    size_t nmatch, regmatch_t pmatch[], int eflags);
size_t regerror(int errcode, const regex_t *preg,
    char *errbuf, size_t errbuf_size);
void regfree(regex_t *preg);
```

Issue 5: No functional changes in this issue.

regex()

Execute a regular expression. (**LEGACY**)

```
#include <libgen.h>

char *regex (const char *re, const char *subject, ...);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations.

advance(), compile(), step(), loc1(), loc2(), locs()

Compile and match regular expressions. (**LEGACY**)

```
#define INIT declarations
#define GETC() getc code
#define PEEK() peek code
#define UNGETC() ungetc code
#define RETURN(ptr) return code
#define ERROR(val) error code

#include <regexp.h>

char *compile(char *instring, char *expbuf,
    const char *endbuf, int eof);
int step(const char *string, const char *expbuf);
int advance(const char *string, const char *expbuf);
extern char *loc1, *loc2, *locs;
```

Issue 5: This function is marked LEGACY and may not be available on all implementations.

A note indicating that these interfaces need not be reentrant is added to the Description. A portable multi-threaded application may only safely call these functions when access to the function is serialized.

remainder( )

Remainder function.

```
#include <math.h>

double remainder(double x, double y);
```

Issue 5: No functional changes in this issue.

remove( )

Remove files.

```
#include <stdio.h>

int remove(const char *path);
```

Issue 5: No functional changes in this issue.

remque( )

Remove an element from a queue.

```
#include <search.h>

void remque(void *element);
```

Issue 5: No functional changes in this issue.

rename( )

Rename a file.

```
#include <stdio.h>

int rename(const char *old, const char *new);
```

Issue 5: The [EBUSY] error is added to the ‘‘may fail’’ part of the Errors section.

rewind( )

Reset the file position indicator in a stream.

```
#include <stdio.h>

void rewind(FILE *stream);
```

Issue 5: No functional changes in this issue.

rewinddir( )

Reset the position of a directory stream to the beginning of a directory.

```
#include <sys/types.h>
#include <dirent.h>

void rewinddir(DIR *dirp);
```

Issue 5: No functional changes in this issue.

rindex( )

Character string operations.

```
#include <strings.h>

char *rindex(const char *s, int c);
```

Issue 5: No functional changes in this issue.

rint( )

Round-to-nearest integral value.

```
#include <math.h>

double rint(double x);
```

Issue 5: No functional changes in this issue.

rmdir( )

Remove a directory.

```
#include <unistd.h>

int rmdir(const char *path);
```

Issue 5: No functional changes in this issue.

sbrk( )

Change space allocation. (**LEGACY**)

```
#include <unistd.h>

void *sbrk(intptr_t incr);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations.

The type of the argument to *sbrk*( ) is changed from **int** to **intptr_t**.

scalb( )

Load an exponent of a radix-independent floating-point number.

```
#include <math.h>

double scalb(double x, double n);
```

Issue 5: No functional changes in this issue.

scanf( )

Convert formatted input.

```
#include <stdio.h>

int scanf(const char *format, ...);
```

Issue 5: No functional changes in this issue.

sched_get_priority_max( ), sched_get_priority_min( )

Get priority limits. (**REALTIME**)

```
#include <sched.h>

int sched_get_priority_max(int policy);
int sched_get_priority_min(int policy);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *sched_get_priority_max*( ) and *sched_get_priority_min*( ) functions return the appropriate maximum or minimum, respectfully, for the scheduling policy specified by *policy*.

sched_getparam( )

Get scheduling parameters. (**REALTIME**)

```
#include <sched.h>

int sched_getparam(pid_t pid, struct sched_param *param);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *sched_getparam*( ) function returns the scheduling parameters of a process specified by *pid* in the **sched_param** structure pointed to by *param*.

sched_getscheduler( )

Get scheduling policy. (**REALTIME**)

```
#include <sched.h>

int sched_getscheduler(pid_t pid);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *sched_getscheduler*( ) function returns the scheduling policy of the process specified by *pid*.

sched_rr_get_interval( )

Get execution time limits. (**REALTIME**)

```
#include <sched.h>

int sched_rr_get_interval(pid_t pid, struct timespec *interval);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *sched_rr_get_interval*( ) function updates the **timespec** structure referenced by the *interval* argument to contain the current execution time limit (that is, time quantum) for the process specified by *pid*.

sched_setparam( )

Set scheduling parameters. (**REALTIME**)

```
#include <sched.h>

int sched_setparam(pid_t pid, const struct sched_param *param);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *sched_setparam*( ) function sets the scheduling parameters of the process specified by *pid* to the values specified by the **sched_param** structure pointed to by *param*.

sched_setscheduler( )

Set scheduling policy and parameters. (**REALTIME**)

```
#include <sched.h>

int sched_setscheduler(pid_t pid, int policy,
    const struct sched_param *param);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *sched_setscheduler*( ) function sets the scheduling policy and scheduling parameters of the process specified by *pid* to *policy* and the parameters specified in the **sched_param** structure pointed to by *param*, respectively.

sched_yield( )

Yield processor.

```
#include <sched.h>

int sched_yield(void);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

The *sched_yield*( ) function forces the running thread to relinquish the processor until it again becomes the head of its thread list.

seed48( )

Seed uniformly distributed pseudo-random non-negative long integer generator.

```
#include <stdlib.h>

unsigned short int *seed48(unsigned short int seed16v[3]);
```

Issue 5: No functional changes in this issue.

seekdir( )

Set position of directory stream.

```
#include <sys/types.h>
#include <dirent.h>

void seekdir(DIR *dirp, long int loc);
```

Issue 5: No functional changes in this issue.

select( )

Synchronous I/O multiplexing.

```
#include <sys/time.h>

int select(int nfds, fd_set *readfds, fd_set *writefds,
    fd_set *errorfds, struct timeval *timeout);
void FD_CLR(int fd, fd_set *fdset);
int FD_ISSET(int fd, fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_ZERO(fd_set *fdset);
```

Issue 5: In the Errors section, the text has been changed to indicate that [EINVAL] will be returned when *nfds* is less than 0 or greater than FD_SETSIZE. It previously stated less than 0, or greater than or equal to FD_SETSIZE.

Text about **timeout** is moved from the Application Usage section to the Description.

sem_close( )

Close a named semaphore. (**REALTIME**)

```
#include <semaphore.h>

int sem_close(sem_t *sem);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *sem_close*( ) function is used to indicate that the calling process is finished using the named semaphore indicated by *sem*.

sem_destroy( )

Destroy an unnamed semaphore. (**REALTIME**)

```
#include <semaphore.h>

int sem_destroy(sem_t *sem);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *sem_destroy*( ) function is used to destroy the unnamed semaphore indicated by *sem*.

sem_getvalue( )

Get the value of a semaphore. (**REALTIME**)

```
#include <semaphore.h>

int sem_getvalue(sem_t *sem, int *sval);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *sem_getvalue*( ) function updates the location referenced by the *sval* argument to have the value of the semaphore referenced by *sem* without affecting the state of the semaphore.

sem_init( )

Initialize an unnamed semaphore. (**REALTIME**)

```
#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, unsigned int value);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *sem_init*( ) function is used to initialize the unnamed semaphore referred to by *sem*.

sem_open( )

Initialize and open a named semaphore. (**REALTIME**)

```
#include <semaphore.h>

sem_t *sem_open(const char *name, int oflag, ...);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *sem_open*( ) function establishes a connection between a named semaphore and a process. Following a call to *sem_open*( ) with semaphore name *name*, the process may reference the semaphore associated with *name* using the address returned from the call.

sem_post( )

Unlock a semaphore. (**REALTIME**)

```
#include <semaphore.h>

int sem_post(sem_t *sem);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *sem_post*( ) function unlocks the semaphore referenced by *sem* by performing a semaphore unlock operation on that semaphore.

sem_unlink( )

Remove a named semaphore. (**REALTIME**)

```
#include <semaphore.h>

int sem_unlink(const char *name);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *sem_unlink*( ) function removes the semaphore named by the string *name*.

sem_wait( ), sem_trywait( )

Lock a semaphore. (**REALTIME**)

```
#include <semaphore.h>

int sem_wait(sem_t *sem);
int sem_trywait(sem_t *sem);
```

Issue 5: These are new functions included for alignment with the POSIX Realtime Extension. These are part of the Realtime Feature Group and may not be available on all implementations.

The *sem_wait*( ) function locks the semaphore referenced by *sem* by performing a semaphore lock operation on that semaphore.

The *sem_trywait*( ) function locks the semaphore referenced by *sem* only if the semaphore is currently not locked; that is, if the semaphore value is currently positive.

semctl( )

Semaphore control operations.

```
#include <sys/sem.h>

int semctl(int semid, int semnum, int cmd, ...);
```

Issue 5: The note about use of POSIX Realtime Extension IPC routines has been moved from Future Directions to the Application Usage section, and reworded as follows:

‘‘The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines can be easily modified to use

the alternative interfaces.''

semget( )

Get set of semaphores.

```
#include <sys/sem.h>

int semget(key_t key, int nsems, int semflg);
```

Issue 5: The note about use of POSIX Realtime Extension IPC routines has been moved from Future Directions to a new Application Usage section, and reworded as follows:

''The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines can be easily modified to use the alternative interfaces.''

semop( )

Semaphore operations.

```
#include <sys/sem.h>

int semop(int semid, struct sembuf *sops, size_t nsops);
```

Issue 5: The note about use of POSIX Realtime Extension IPC routines has been moved from Future Directions to a new Application Usage section, and reworded as follows:

''The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines can be easily modified to use the alternative interfaces.''

setbuf( )

Assign buffering to a stream.

```
#include <stdio.h>

void setbuf(FILE *stream, char *buf);
```

Issue 5: No functional changes in this issue.

setcontext( )

Set current user context.

```
#include <ucontext.h>

int setcontext(const ucontext_t *ucp);
```

Issue 5: No functional changes in this issue.

setgid( )

Set-group-ID.

```
#include <sys/types.h>
#include <unistd.h>

int setgid(gid_t gid);
```

Issue 5: No functional changes in this issue.

setgrent( )

Reset group database to the first entry.

```
#include <grp.h>

void setgrent(void);
```

Issue 5: No functional changes in this issue.

setitimer( )

Set the value of the interval timer.

```
#include <sys/time.h>

int setitimer(int which, const struct itimerval *value,
    struct itimerval *ovalue);
```

Issue 5: No functional changes in this issue.

_setjmp( )

Set the jump point for a non-local goto.

```
#include <setjmp.h>

int _setjmp(jmp_buf env);
```

Issue 5: No functional changes in this issue.

setjmp( )

Set the jump point for a non-local goto.

```
#include <setjmp.h>

int setjmp(jmp_buf env);
```

Issue 5: No functional changes in this issue.

setkey( )

Set the encoding key. (**CRYPT**)

```
#include <stdlib.h>

void setkey(const char *key);
```

Issue 5: The Description is updated to indicate that *errno* will not be changed if the function is successful.

setlocale( )

Set the program locale.

```
#include <locale.h>

char *setlocale(int category, const char *locale);
```

Issue 5: The Description is updated for alignment with the POSIX Threads Extension. Specifically, the locale state is now described as common to all threads within a process.

setlogmask( )

Set the log priority mask.

```
#include <syslog.h>

int setlogmask(int maskpri);
```

Issue 5: No functional changes in this issue.

setpgid( )

Set the process group ID for job control.

```
#include <sys/types.h>
#include <unistd.h>

int setpgid(pid_t pid, pid_t pgid);
```

Issue 5: No functional changes in this issue.

setpgrp( )

Set the process group ID.

```
#include <unistd.h>

pid_t setpgrp(void);
```

Issue 5: No functional changes in this issue.

setpriority( )

Set the nice value.

```
#include <sys/resource.h>

int setpriority(int which, id_t who, int nice);
```

Issue 5: The *nice* value is added.

setpwent( )

User database function.

```
#include <pwd.h>

void setpwent(void);
```

Issue 5: No functional changes in this issue.

setregid( )

Set real and effective group IDs.

```
#include <unistd.h>

int setregid(gid_t rgid, gid_t egid);
```

Issue 5: The Description is updated to indicate that the saved set-group-ID can be set by any of the *exec**( ) functions, not just *execv*( ).

setreuid( )

Set real and effective user IDs.

```
#include <unistd.h>

int setreuid(uid_t ruid, uid_t euid);
```

Issue 5: No functional changes in this issue.

setrlimit( )

Control maximum resource consumption.

```
#include <sys/resource.h>

int setrlimit(int resource, const struct rlimit *rlp);
```

Issue 5: No functional changes in this issue.

setsid( )

Create session and set process group ID.

```
#include <sys/types.h>
#include <unistd.h>

pid_t setsid(void);
```

Issue 5: No functional changes in this issue.

setstate( )

Switch pseudo-random number generator state arrays.

```
#include <stdlib.h>

char *setstate(const char *state);
```

Issue 5: No functional changes in this issue.

setuid( )

Set-user-ID.

```
#include <sys/types.h>
#include <unistd.h>

int setuid(uid_t uid);
```

Issue 5: No functional changes in this issue.

setutxent( )

Reset user accounting database to first entry.

```
#include <utmpx.h>

void setutxent(void);
```

Issue 5: No functional changes in this issue.

setvbuf( )

Assign buffering to a stream.

```
#include <stdio.h>

int setvbuf(FILE *stream, char *buf, int type, size_t size);
```

Issue 5: No functional changes in this issue.

shm_open( )

Open a shared memory object. (**REALTIME**)

```
#include <sys/mman.h>

int shm_open(const char *name, int oflag, mode_t mode);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *shm_open*( ) function establishes a connection between a shared memory object and a file descriptor.

shm_unlink( )

Remove a shared memory object. (**REALTIME**)

```
#include <sys/mman.h>

int shm_unlink(const char * name);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *shm_unlink*( ) function removes the name of the shared memory object named by the string pointed to by *name*.

shmat( )

Shared memory attach operation.

```
#include <sys/shm.h>

void *shmat(int shmid, const void *shmaddr, int shmflg);
```

Issue 5: The note about use of POSIX Realtime Extension IPC routines has been moved from Future Directions to a new Application Usage section, and reworded as follows:

‘‘The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their

applications so that modules using the IPC routines can be easily modified to use the alternative interfaces.''

shmctl( )

Shared memory control operations.

```
#include <sys/shm.h>

int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

Issue 5: The note about use of POSIX Realtime Extension IPC routines has been moved from Future Directions to a new Application Usage section, and reworded as follows:

''The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines can be easily modified to use the alternative interfaces.''

shmdt( )

Shared memory detach operation.

```
#include <sys/shm.h>

int shmdt(const void *shmaddr);
```

Issue 5: The note about use of POSIX Realtime Extension IPC routines has been moved from Future Directions to a new Application Usage section, and reworded as follows:

''The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines can be easily modified to use the alternative interfaces.''

shmget( )

Get shared memory segment.

```
#include <sys/shm.h>

int shmget(key_t key, size_t size, int shmflg);
```

Issue 5: The note about use of POSIX Realtime Extension IPC routines has been moved from Future Directions to a new Application Usage section, and reworded as follows:

''The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines can be easily modified to use the alternative interfaces.''

sigaction( )

Examine and change signal action.

```
#include <signal.h>

int sigaction(int sig, const struct sigaction *act,
    struct sigaction *oact);
```

Issue 5: The Description is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

In the Description, the second argument to *func*( ) when SA_SIGINFO is set is no longer permitted to be NULL, and the description of permitted **siginfo_t** contents is expanded by reference to **<signal.h>**.

Because the X/Open UNIX Extension functionality is now folded into the BASE, the [ENOTSUP] error is deleted.

sigaddset( )

Add a signal to a signal set.

```
#include <signal.h>

int sigaddset(sigset_t *set, int signo);
```

Issue 5: The last paragraph of the Description was included as an Application Usage note in previous issues. This states:

‘‘Applications must call either *sigemptyset*( ) or *sigfillset*( ) at least once for each object of type **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is nonetheless supplied as an argument to any of *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigismember*( ), *sigpending*( ), or *sigprocmask*( ), the results are undefined.

sigaltstack( )

Set and/or get signal alternate stack context.

```
#include <signal.h>

int sigaltstack(const stack_t *ss, stack_t *oss);
```

Issue 5: The last sentence of the Description was included as an Application Usage note in previous issues. This states:

‘‘In some implementations, a signal (whether or not indicated to execute on the alternate stack) will always execute on the alternate stack if it is delivered while another signal is being caught using the alternate stack.’’

The following is also added:

‘‘Use of this function by library threads that are not bound to kernel-scheduled entities results in undefined behavior.’’

sigdelset( )

Delete a signal from a signal set.

```
#include <signal.h>

int sigdelset(sigset_t *set, int signo);
```

Issue 5: The last paragraph of the Description was included as an Application Usage note in previous issues. This states:

‘‘Applications should call either *sigemptyset*( ) or *sigfillset*( ) at least once for each object of type **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is nonetheless supplied as an argument to any of *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigismember*( ), *sigpending*( ), or *sigprocmask*( ), the results are undefined.

sigemptyset( )

Initialize and empty a signal set.

```
#include <signal.h>

int sigemptyset(sigset_t *set);
```

Issue 5: No functional changes in this issue.

sigfillset( )

Initialize and fill a signal set.

```
#include <signal.h>

int sigfillset(sigset_t *set);
```

Issue 5: No functional changes in this issue.

sighold( ), sigignore( )

Add a signal to the signal mask or set a signal disposition to be ignored.

```
#include <signal.h>

int sighold(int sig);
int sigignore(int sig);
```

Issue 5: No functional changes in this issue.

siginterrupt( )

Allow signals to interrupt functions.

```
#include <signal.h>

int siginterrupt(int sig, int flag);
```

Issue 5: No functional changes in this issue.

sigismember( )

Test for a signal in a signal set.

```
#include <signal.h>

int sigismember(const sigset_t *set, int signo);
```

Issue 5: The last paragraph of the Description was included as an Application Usage note in previous issues. This states:

‘‘Applications should call either *sigemptyset*( ) or *sigfillset*( ) at least once for each object of type **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is nonetheless supplied as an argument to any of *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigismember*( ), *sigpending*( ), or *sigprocmask*( ), the results are undefined.

siglongjmp( )

Non-local goto with signal handling.

```
#include <setjmp.h>

void siglongjmp(sigjmp_buf env, int val);
```

Issue 5: The Description is updated for alignment with the POSIX Threads Extension. Specifically, this is now described in terms of a thread rather than a process, and the following statement added:

‘‘The effect of a call to *siglongjmp*( ), where initialization of the **jmp_buf** structure was not performed in the calling thread, is undefined.’’

signal( ), sigset( ), sighold( ), sigrelse( ), sigignore( ), sigpause( )

Signal management.

```
#include <signal.h>

void (*signal(int sig, void (*func)(int)))(int);
int sighold(int sig);
int sigignore(int sig);
int sigpause(int sig);
int sigrelse(int sig);
void (*sigset(int sig, void (*disp)(int)))(int);
```

Issue 5: The Description is updated to indicate that the *sigpause*( ) function restores the process’ signal mask to its original state before returning.

The Return Value section is updated to indicate that the *sigpause*( ) function suspends execution of the process until a signal is received, whereupon it returns −1 and sets *errno* to [EINTR].

signgam( )

Storage for sign of *lgamma*( ).

```
#include <math.h>

extern int signgam;
```

Issue 5: No functional changes in this issue.

sigpause( )

Remove a signal from the signal mask and suspend the thread.

```
#include <signal.h>

int sigpause(int sig);
```

Issue 5: No functional changes in this issue.

sigpending( )

Examine pending signals.

```
#include <signal.h>

int sigpending(sigset_t *set);
```

Issue 5: The Description is updated for alignment with the POSIX Threads Extension, specifically this now states:

''The *sigpending*( ) function stores, in the location referenced by the *set* argument, the set of signals that are blocked from delivery to the calling thread and that are pending on the process or the calling thread.''

sigprocmask( ), pthread_sigmask( )

Examine and change blocked signals.

```
#include <signal.h>

int sigprocmask(int how, const sigset_t *set, sigset_t *oset);
int pthread_sigmask(int how, const sigset_t *set, sigset_t *oset);
```

Issue 5: The Description is updated for alignment with the POSIX Threads Extension. Specifically, the following changes are made:

- The description is updated to include threads as well as processes.
- The use of the *sigprocmask*( ) function is unspecified in a multi-threaded process.
- If *sigprocmask*( ) fails, the thread's signal mask is not changed.
- The *pthread_sigmask*( ) function is new and is used to examine or change (or both) the calling thread's signal mask, regardless of the number of threads in the process. The effect is the same as described for *sigprocmask*( ), without the restriction that the call be made in a single-threaded process.

sigqueue( )

Queue a signal to a process. (**REALTIME**)

```
#include <sys/types.h>
#include <signal.h>

int sigqueue(pid_t pid, int signo, const union sigval value);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations and the POSIX Threads Extension.

The *sigqueue*( ) function causes the signal specified by *signo* to be sent with the value specified by *value* to the process specified by *pid*.

sigrelse( ), sigset( )

Remove a signal from the signal mask or modify the signal disposition.

```
#include <signal.h>

int sigrelse(int sig);
void (*sigset(int sig, void (*disp)(int)))(int);
```

Issue 5: No functional changes in this issue.

sigsetjmp( )

Set the jump point for a non-local goto.

```
#include <setjmp.h>

int sigsetjmp(sigjmp_buf env, int savemask);
```

Issue 5: The Description is updated for alignment with the POSIX Threads Extension. Specifically, this now describes the signal mask for the calling thread rather than the process.

sigstack( )

Set and/or get an alternate signal stack context. (**LEGACY**)

```
#include <signal.h>

int sigstack(struct sigstack *ss, struct sigstack *oss);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations.

A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *sigstack*( ) when access to the function is serialized.

A portable application, when being written or rewritten, should use *sigaltstack*( ) instead of *sigstack*( ).

sigsuspend( )

Wait for a signal.

```
#include <signal.h>

int sigsuspend(const sigset_t *sigmask);
```

Issue 5: The Description is updated for alignment with the POSIX Threads Extension. The first paragraph is reworded as follows:

''The *sigsuspend*( ) function replaces the current signal mask of the calling thread with the set of signals pointed to by *sigmask* and then suspends the thread until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process. This will not cause any other signals that may have been pending on the process to become pending on the thread.''

sigwait( )

Wait for queued signals.

```
#include <signal.h>

int sigwait(const sigset_t *set, int *sig);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

The *sigwait*( ) function selects a pending signal from *set*, atomically clears it from the system's set of pending signals, and returns that signal number in the location referenced by *sig*.

sigwaitinfo( ), sigtimedwait( )

Wait for queued signals. (**REALTIME**)

```
#include <signal.h>

int sigwaitinfo(const sigset_t *set, siginfo *info);
int sigtimedwait(const sigset_t *set, siginfo_t *info,
    const struct timespec *timeout);
```

Issue 5: These are new functions included for alignment with the POSIX Realtime Extension. These are part of the Realtime Feature Group and may not be available on all implementations.

The function *sigwaitinfo*( ) selects the pending signal from the set specified by *set*.

The function *sigwaitinfo*( ) behaves the same as the *sigwait*( ) function if the *info* argument is NULL.

sin( )

Sine function.

```
#include <math.h>

double sin(double x);
```

Issue 5: The last two paragraphs of the Description were included as Application Usage notes in previous issues. This describes how applications should check for an error. Note how *sin*( ) may lose accuracy when its argument is far from 0.0.

sinh()

Hyperbolic sine function.

```
#include <math.h>

double sinh(double x);
```

Issue 5: No functional changes are made in this issue, although the Description is updated to indicate how an application should check for an error. This text was previously published in the Application Usage section.

sleep()

Suspend execution for an interval of time.

```
#include <unistd.h>

unsigned int sleep(unsigned int seconds);
```

Issue 5: The Description is updated for alignment with the POSIX Threads Extension. Specifically, this describes the calling thread being suspended, rather than the process. Interactions between *sleep*() and any of *setitimer*(), *ualarm*(), or *usleep*() are unspecified.

sprintf(), snprintf()

Print formatted output.

```
#include <stdio.h>

int snprintf(char *s, size_t n, const char *format, /* args */ ...);
int sprintf(char *s, const char *format, ...);
```

Issue 5: The *snprintf*() function is new in Issue 5. See *fprintf*().

sqrt()

Square root function.

```
#include <math.h>

double sqrt(double x);
```

Issue 5: No functional changes in this issue.

srand()

Seed simple pseudo-random number generator.

```
#include <stdlib.h>

void srand(unsigned int seed);
```

Issue 5: No functional changes in this issue.

srand48( )

Seed uniformly distributed double-precision pseudo-random number generator.

```
#include <stdlib.h>

void srand48(long int seedval);
```

Issue 5: No functional changes in this issue.

srandom( )

Seed pseudo-random number generator.

```
#include <stdlib.h>

void srandom(unsigned int seed);
```

Issue 5: No functional changes in this issue.

sscanf( )

Convert formatted input.

```
#include <stdio.h>

int sscanf(const char *s, const char *format, ...);
```

Issue 5: No functional changes in this issue.

stat( )

Get file status.

```
#include <sys/types.h>
#include <sys/stat.h>

int stat(const char *path, struct stat *buf);
```

Issue 5: Large File Support extensions are added; see Chapter 17 on page 163.

statvfs( )

Get filesystem information.

```
#include <sys/statvfs.h>

int statvfs(const char *path, struct statvfs *buf);
```

Issue 5: No functional changes in this issue.

stderr, stdin, stdout

Standard I/O streams.

```
#include <stdio.h>

extern FILE *stderr, *stdin, *stdout;
```

Issue 5: No functional changes in this issue.

step( )

Pattern match with regular expressions. (**LEGACY**)

```
#include <regexp.h>

int step(const char *string, const char *expbuf);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations.

strcasecmp( ), strncasecmp( )

Case-insensitive string comparisons.

```
#include <strings.h>

int strcasecmp(const char *s1, const char *s2);
int strncasecmp(const char *s1, const char *s2, size_t n);
```

Issue 5: No functional changes in this issue.

strcat( )

Concatenate two strings.

```
#include <string.h>

char *strcat(char *s1, const char *s2);
```

Issue 5: No functional changes in this issue.

strchr( )

String scanning operation.

```
#include <string.h>

char *strchr(const char *s, int c);
```

Issue 5: No functional changes in this issue.

strcmp( )

Compare two strings.

```
#include <string.h>

int strcmp(const char *s1, const char *s2);
```

Issue 5: No functional changes in this issue.

strcoll( )

String comparison using collating information.

```
#include <string.h>

int strcoll(const char *s1, const char *s2);
```

Issue 5: The Description is updated to indicate that *errno* will not be changed if the function is successful.

strcpy( )

Copy a string.

```
#include <string.h>

char *strcpy(char *s1, const char *s2);
```

Issue 5: No functional changes in this issue.

strcspn( )

Get the length of a complementary substring.

```
#include <string.h>

size_t strcspn(const char *s1, const char *s2);
```

Issue 5: The Return Value section is updated to indicated that *strcspn*( ) returns the length of *s1*, and not *s1* itself as was previously stated.

strdup( )

Duplicate a string.

```
#include <string.h>

char *strdup(const char *s1);
```

Issue 5: No functional changes in this issue.

strerror( )

Get an error message string.

```
#include <string.h>

char *strerror(int errnum);
```

Issue 5: The Description is updated to indicate that *errno* will not be changed if the function is successful.

A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *strerror*( ) when access to the function is serialized.

strfmon( )

Convert a monetary value to a string.

```
#include <monetary.h>

ssize_t strfmon(char *s, size_t maxsize, const char *format, ...);
```

Issue 5: A sentence is added to the Description warning about values of *maxsize* that are greater than {SSIZE_MAX}.

strftime( )

Convert date and time to a string.

```
#include <time.h>

size_t strftime(char *s, size_t maxsize, const char *format,
    const struct tm *timptr);
```

Issue 5: The description of %*OV* is changed to be consistent with %*V* and defines Monday as the first day of the week.

The description of %*Oy* is clarified:

%*Oy* Replaced by the year (offset from %*C*) using the locale's alternative numeric symbols.

strlen( )

Get string length.

```
#include <string.h>

size_t strlen(const char *s);
```

Issue 5: The Return Value section is updated to indicate that *strlen*( ) returns the length of *s*, and not *s* itself as was previously stated.

strncasecmp( )

Case-insensitive string comparison.

```
#include <strings.h>

int strncasecmp(const char *s1, const char *s2, size_t n);
```

Issue 5: No functional changes in this issue.

strncat( )

Concatenate part of two strings.

```
#include <string.h>

char *strncat(char *s1, const char *s2, size_t n);
```

Issue 5: No functional changes in this issue.

strncmp( )

Compare part of two strings.

```
#include <string.h>

int strncmp(const char *s1, const char *s2, size_t n);
```

Issue 5: No functional changes in this issue.

strncpy( )

Copy part of a string.

```
#include <string.h>

char *strncpy(char *s1, const char *s2, size_t n);
```

Issue 5: No functional changes in this issue.

strpbrk( )

Scan a string for a byte.

```
#include <string.h>

char *strpbrk(const char *s1, const char *s2);
```

Issue 5: No functional changes in this issue.

strptime( )

Date and time conversion.

```
#include <time.h>

char *strptime(const char *buf, const char *format, struct tm *tm);
```

Issue 5: The exact meaning of the %*y* and %*Oy* specifiers is clarified in the Description:

%*y* The year within century. When a century is not otherwise specified, values in the range 69-99 refer to years in the twentieth century (1969 to 1999 inclusive); values in the range 00-68 refer to years in the twenty-first century (2000 to 2068 inclusive). Leading zeros are permitted but not required.

%*Oy* The year (offset from %*C*) using the locale's alternative numeric symbols.

strrchr( )

String scanning operation.

```
#include <string.h>

char *strrchr(const char *s, int c);
```

Issue 5: No functional changes in this issue.

strspn( )

Get length of a substring.

```
#include <string.h>

size_t strspn(const char *s1, const char *s2);
```

Issue 5: The Return Value section is updated to indicate that *strspn*( ) returns the length of *s*, and not *s* itself as was previously stated.

strstr( )

Find a substring.

```
#include <string.h>

char *strstr(const char *s1, const char *s2);
```

Issue 5: No functional changes in this issue.

strtod( )

Convert a string to a double-precision number.

```
#include <stdlib.h>

double strtod(const char *str, char **endptr);
```

Issue 5: The Description is updated to indicate that *errno* will not be changed if the function is successful.

strtok( ), strtok_r( )

Split a string into tokens.

```
#include <string.h>

char *strtok(char *s1, const char *s2);
char *strtok_r(char *s, const char *sep, char **lasts);
```

Issue 5: The *strtok_r*( ) function is included for alignment with the POSIX Threads Extension, and is the thread-safe equivalent of *strtok*( ).

A note indicating that the *strtok*( ) interface need not be reentrant is added to the Description. A portable multi-threaded application should call *strtok_r*( ) instead of *strtok*( ).

strtol( )

Convert a string to a long integer.

```
#include <stdlib.h>

long int strtol(const char *str, char **endptr, int base);
```

Issue 5: The Description is updated to indicate that *errno* will not be changed if the function is successful.

strtoul( )

Convert a string to an unsigned long.

```
#include <stdlib.h>

unsigned long int strtoul(const char *str, char **endptr, int base);
```

Issue 5: The Description is updated to indicate that *errno* will not be changed if the function is successful.

strxfrm( )

String transformation.

```
#include <string.h>

size_t strxfrm(char *s1, const char *s2, size_t n);
```

Issue 5: The Description is updated to indicate that *errno* will not be changed if the function is successful.

swab( )

Swap bytes.

```
#include <unistd.h>

void swab(const void *src, void *dest, ssize_t nbytes);
```

Issue 5: No functional changes in this issue.

swapcontext( )

Swap user context.

```
#include <ucontext.h>

int swapcontext(ucontext_t *oucp, const ucontext_t *ucp);
```

Issue 5: No functional changes in this issue.

swprintf( )

Print formatted wide-character output.

```
#include <stdio.h>
#include <wchar.h>

int swprintf(wchar_t *s, size_t n, const wchar_t *format, ...);
```

Issue 5: Included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The is the wide-character equivalent of *sprintf*( ).

swscanf( )

Convert formatted wide-character input.

```
#include <stdio.h>
#include <wchar.h>

int swscanf(const wchar_t *s, const wchar_t *format, ...);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The is the wide-character equivalent of *sscanf*( ).

symlink( )

Make a symbolic link to a file.

```
#include <unistd.h>

int symlink(const char *path1, const char *path2);
```

Issue 5: No functional changes in this issue.

sync( )

Schedule filesystem updates.

```
#include <unistd.h>

void sync(void);
```

Issue 5: No functional changes in this issue.

sysconf( )

Get configurable system variables.

```
#include <unistd.h>

long int sysconf(int name);
```

Issue 5: The Description is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension. Specifically, the following variables and name values are added:

Variable	Value of Name
AIO_LISTIO_MAX	_SC_AIO_LISTIO_MAX
AIO_MAX	_SC_AIO_MAX
AIO_PRIO_DELTA_MAX	_SC_AIO_PRIO_DELTA_MAX
DELAYTIMER_MAX	_SC_DELAYTIMER_MAX
MQ_OPEN_MAX	_SC_MQ_OPEN_MAX
MQ_PRIO_MAX	_SC_MQ_PRIO_MAX
RTSIG_MAX	_SC_RTSIG_MAX
SEM_NSEMS_MAX	_SC_SEM_NSEMS_MAX
SEM_VALUE_MAX	_SC_SEM_VALUE_MAX
SIGQUEUE_MAX	_SC_SIGQUEUE_MAX
TIMER_MAX	_SC_TIMER_MAX
_POSIX_ASYNCHRONOUS_IO	_SC_ASYNCHRONOUS_IO
_POSIX_FSYNC	_SC_FSYNC
_POSIX_MAPPED_FILES	_SC_MAPPED_FILES
_POSIX_MEMLOCK	_SC_MEMLOCK
_POSIX_MEMLOCK_RANGE	_SC_MEMLOCK_RANGE
_POSIX_MEMORY_PROTECTION	_SC_MEMORY_PROTECTION
_POSIX_MESSAGE_PASSING	_SC_MESSAGE_PASSING
_POSIX_PRIORITIZED_IO	_SC_PRIORITIZED_IO
_POSIX_PRIORITY_SCHEDULING	_SC_PRIORITY_SCHEDULING
_POSIX_REALTIME_SIGNALS	_SC_REALTIME_SIGNALS
_POSIX_SEMAPHORES	_SC_SEMAPHORES
_POSIX_SHARED_MEMORY_OBJECTS	_SC_SHARED_MEMORY_OBJECTS
_POSIX_SYNCHRONIZED_IO	_SC_SYNCHRONIZED_IO

Variable	Value of Name
_POSIX_TIMERS	_SC_TIMERS
Maximum size of *getgrgid_r*( ) and *getgrnam_r*( ) data buffers	_SC_GETGR_R_SIZE_MAX
Maximum size of *getpwuid_r*( ) and *getpwnam_r*( ) data buffers	_SC_GETPW_R_SIZE_MAX
LOGIN_NAME_MAX	_SC_LOGIN_NAME_MAX
PTHREAD_DESTRUCTOR_ITERATIONS	_SC_THREAD_DESTRUCTOR_ ITERATIONS
PTHREAD_KEYS_MAX	_SC_THREAD_KEYS_MAX
PTHREAD_STACK_MIN	_SC_THREAD_STACK_MIN
PTHREAD_THREADS_MAX	_SC_THREAD_THREADS_MAX
TTY_NAME_MAX	_SC_TTY_NAME_MAX
_POSIX_THREADS	_SC_THREADS
_POSIX_THREAD_ATTR_STACKADDR	_SC_THREAD_ATTR_STACKADDR
_POSIX_THREAD_ATTR_STACKSIZE	_SC_THREAD_ATTR_STACKSIZE
_POSIX_THREAD_PRIORITY_ SCHEDULING	_SC_THREAD_PRIORITY_SCHEDULING
_POSIX_THREAD_PRIO_INHERIT	_SC_THREAD_PRIO_INHERIT
_POSIX_THREAD_PRIO_PROTECT	_SC_THREAD_PRIO_PROTECT
_POSIX_THREAD_PROCESS_SHARED	_SC_THREAD_PROCESS_SHARED
_POSIX_THREAD_SAFE_FUNCTIONS	_SC_THREAD_SAFE_FUNCTIONS

and the new Feature Groups and data size neutrality values are added to the table of system variables:

Variable	Value of Name
_XBS5_ILP32_OFF32	_SC_XBS5_ILP32_OFF32
_XBS5_ILP32_OFFBIG	_SC_XBS5_ILP32_OFFBIG
_XBS5_LP64_OFF64	_SC_XBS5_LP64_OFF64
_XBS5_LPBIG_OFFBIG	_SC_XBS5_LPBIG_OFFBIG
_XOPEN_REALTIME	_SC_XOPEN_REALTIME
_XOPEN_REALTIME_THREADS	_SC_XOPEN_REALTIME_THREADS
_XOPEN_LEGACY	_XOPEN_LEGACY

syslog( )

Log a message.

```
#include <syslog.h>

void syslog(int priority, const char *message, ... /* argument */);
```

Issue 5: No functional changes in this issue.

system( )

Issue a command.

```
#include <stdlib.h>

int system(const char *command);
```

Issue 5: No functional changes in this issue.

tan( )

Tangent function.

```
#include <math.h>

double tan(double x);
```

Issue 5: The last two paragraphs of the Description were included as Application Usage notes in previous issues.

tanh( )

Hyperbolic tangent function.

```
#include <math.h>

double tanh(double x);
```

Issue 5: No functional changes in this issue.

tcdrain( )

Wait for transmission of output.

```
#include <termios.h>

int tcdrain(int fildes);
```

Issue 5: No functional changes in this issue.

tcflow( )

Suspend or restart the transmission or reception of data.

```
#include <termios.h>

int tcflow(int fildes, int action);
```

Issue 5: No functional changes in this issue.

tcflush( )

Flush non-transmitted output data, non-read input data, or both.

```
#include <termios.h>

int tcflush(int fildes, int queue_selector);
```

Issue 5: No functional changes in this issue.

tcgetattr( )

Get the parameters associated with the terminal.

```
#include <termios.h>

int tcgetattr(int fildes, struct termios *termios_p);
```

Issue 5: No functional changes in this issue.

tcgetpgrp( )

Get the foreground process group ID.

```
#include <sys/types.h>
#include <unistd.h>

pid_t tcgetpgrp(int fildes);
```

Issue 5: No functional changes in this issue.

tcgetsid( )

Get the process group ID for the session leader for the controlling terminal.

```
#include <termios.h>

pid_t tcgetsid(int fildes);
```

Issue 5: The [EACCES] error has been removed from the list of mandatory errors, and the description of [ENOTTY] has been reworded.

tcsendbreak( )

Send a break for a specific duration.

```
#include <termios.h>

int tcsendbreak(int fildes, int duration);
```

Issue 5: No functional changes in this issue.

tcsetattr( )

Set the parameters associated with the terminal.

```
#include <termios.h>

int tcsetattr(int fildes, int optional_actions,
    const struct termios *termios_p);
```

Issue 5: No functional changes in this issue.

tcsetpgrp( )

Set the foreground process group ID.

```
#include <sys/types.h>
#include <unistd.h>

int tcsetpgrp(int fildes, pid_t pgid_id);
```

Issue 5: No functional changes in this issue.

tdelete( )

Delete a node from the binary search tree.

```
#include <search.h>

void *tdelete(const void *key, void **rootp,
    int (*compar)(const void *, const void *));
```

Issue 5: No functional changes in this issue.

telldir( )

Current location of a named directory stream.

```
#include <dirent.h>

long int telldir(DIR *dirp);
```

Issue 5: No functional changes in this issue.

tempnam( )

Create a name for a temporary file.

```
#include <stdio.h>

char *tempnam(const char *dir, const char *pfx);
```

Issue 5: The last paragraph of the Description was included as an Application Usage note in previous issues. This states:

> ‘‘Some implementations of *tempnam*( ) may use *tmpnam*( ) internally. On such implementations, if called more than {TMP_MAX} times in a single process, the behavior is implementation-dependent.’’

tfind( )

Search a binary search tree.

```
#include <search.h>

void *tfind(const void *key, void *const *rootp,
    int (*compar)(const void *, const void *));
```

Issue 5: No functional changes in this issue.

time( )

Get time.

```
#include <time.h>

time_t time(time_t *tloc);
```

Issue 5: No functional changes in this issue.

timer_create( )

Create a per-process timer. (**REALTIME**)

```
#include <time.h>
#include <signal.h>

int timer_create(clockid_t clockid, struct sigevent *evp,
    timer_t *timerid);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *timer_create*( ) function creates a per-process timer using the specified clock, *clock_id*, as the timing base.

timer_delete( )

Delete a per-process timer. (**REALTIME**)

```
#include <time.h>

int timer_delete(timer_t timerid);
```

Issue 5: This is a new function included for alignment with the POSIX Realtime Extension. This is part of the Realtime Feature Group and may not be available on all implementations.

The *timer_delete*( ) function deletes the specified timer, *timerid*, previously created by the *timer_create*( ) function.

timer_settime( ), timer_gettime( ), timer_getoverrun( )

Per-process timers. (**REALTIME**)

```
#include <time.h>

int timer_settime(timer_t timerid, int flags,
    const struct itimerspec *value, struct itimerspec *ovalue);
int timer_gettime(timer_t timerid, struct itimerspec *value);
int timer_getoverrun(timer_t timerid);
```

Issue 5: These are new functions included for alignment with the POSIX Realtime Extension. These are part of the Realtime Feature Group and may not be available on all implementations.

The *timer_settime*( ) function sets the time until the next expiration of the timer specified by *timerid* from the *it_value* member of the *value* argument and arms the timer if the *it_value* member of *value* is non-zero.

The *timer_gettime*( ) function stores the amount of time until the specified timer, *timerid*, expires and the reload value of the timer into the space pointed to by the *value* argument.

The *timer_getoverrun*( ) function returns the timer expiration overrun count for the specified timer, when a timer expiration signal occurs for the process.

times( )

Get a process and waited-for child process times.

```
#include <sys/times.h>

clock_t times(struct tms *buffer);
```

Issue 5: No functional changes in this issue.

timezone( )

Difference from UTC and local standard time.

```
#include <time.h>

extern long int timezone;
```

Issue 5: No functional changes in this issue.

tmpfile( )

Create a temporary file.

```
#include <stdio.h>

FILE *tmpfile(void);
```

Issue 5: Large File Support extensions are added; see Chapter 17 on page 163.

The last two paragraphs of the Description were included as Application Usage notes in previous issues.

tmpnam( )

Create a name for a temporary file.

```
#include <stdio.h>

char *tmpnam(char *s);
```

Issue 5: The Description is updated for alignment with the POSIX Threads Extension. Specifically, this adds the following statements:

The implementation will behave as if no function defined in XSH, Issue 5 calls *tmpnam*( ).

If the application uses any of the interfaces guaranteed to be available if either _POSIX_THREAD_SAFE_FUNCTIONS or _POSIX_THREADS is defined, the *tmpnam*( ) function must be called with a non-NULL parameter.

toascii( )

Translate an integer to a 7-bit ASCII character.

```
#include <ctype.h>

int toascii(int c);
```

Issue 5: No functional changes in this issue.

_tolower( )

Transliterate uppercase characters to lowercase.

```
#include <ctype.h>
int _tolower(int c);
```

Issue 5: No functional changes in this issue.

tolower( )

Transliterate uppercase characters to lowercase.

```
#include <ctype.h>
int tolower(int c);
```

Issue 5: No functional changes in this issue.

_toupper( )

Transliterate lowercase characters to uppercase.

```
#include <ctype.h>
int _toupper(int c);
```

Issue 5: No functional changes in this issue.

toupper( )

Transliterate lowercase characters to uppercase.

```
#include <ctype.h>
int toupper(int c);
```

Issue 5: No functional changes in this issue.

towctrans( )

Character transliteration.

```
#include <wctype.h>
wint_t towctrans(wint_t wc, wctrans_t desc);
```

Issue 5: This is a new function derived from the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The *towctrans*() function transliterates the wide-character code *wc* using the mapping described by *desc*.

towlower( )

Transliterate uppercase wide-character code to lowercase.

```
#include <wctype.h>
wint_t towlower(wint_t wc);
```

Issue 5: The following change has been made in this issue for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E):

The Synopsis has been changed to indicate that this function and associated data types are now made visible by inclusion of the header **<wctype.h>** rather than **<wchar.h>**.

towupper( )

Transliterate lowercase wide-character code to uppercase.

```
#include <wctype.h>

wint_t towupper(wint_t wc);
```

Issue 5: The following change has been made in this issue for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E):

The Synopsis has been changed to indicate that this function and associated data types are now made visible by inclusion of the header **<wctype.h>** rather than **<wchar.h>**.

truncate( )

Truncate a file to a specified length.

```
#include <unistd.h>

int truncate(const char *path, off_t length);
```

Issue 5: No functional changes in this issue.

tdelete( ), tfind( ), tsearch( ), twalk( )

Manage a binary search tree.

```
#include <search.h>

void *tsearch(const void *key, void **rootp,
     int (*compar)(const void *, const void *));
void *tfind(const void *key, void *const *rootp,
     int(*compar)(const void *, const void *));
void *tdelete(const void *key, void **rootp,
     int(*compar)(const void *, const void *));
void twalk(const void *root,
     void (*action)(const void *, VISIT, int));
```

Issue 5: The last paragraph of the Description was included as an Application Usage note in previous issues. This states:

''If the calling function alters the pointer to the root, the result is undefined.''

ttyname( ), ttyname_r( )

Find the pathname of a terminal.

```
#include <unistd.h>

char *ttyname(int fildes);
int ttyname_r(int fildes, char *name, size_t namesize);
```

Issue 5: The *ttyname_r*( ) function is included for alignment with the POSIX Threads Extension, and is the thread-safe equivalent of *ttyname*( ).

A note indicating that the *ttyname*( ) interface need not be reentrant is added to the Description. A portable multi-threaded application should call *ttyname_r*( ) instead of *ttyname*( ).

ttyslot( )

Find the slot of the current user in the user accounting database. (**LEGACY**)

```
#include <stdlib.h>

int ttyslot(void);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations.

A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *ttyslot*( ) when access to the function is serialized.

twalk( )

Traverse a binary search tree.

```
#include <search.h>

void twalk(const void *root,
    void (*action)(const void *, VISIT, int ));
```

Issue 5: No functional changes in this issue.

tzname( )

Timezone strings.

```
#include <time.h>

extern char *tzname[];
```

Issue 5: No functional changes in this issue.

tzset( )

Set timezone conversion information.

```
#include <time.h>

void tzset (void);
extern char *tzname[];
extern long int timezone;
extern int daylight;
```

Issue 5: No functional changes in this issue.

ualarm( )

Set the interval timer.

```
#include <unistd.h>

useconds_t ualarm(useconds_t useconds, useconds_t interval);
```

Issue 5: No functional changes in this issue.

ulimit( )

Get and set process limits.

```
#include <ulimit.h>

long int ulimit(int cmd, ...);
```

Issue 5: In the description of UL_SETFSIZE, the text is corrected to refer to **rlim_t** rather than the spurious **rlimit_t**.

The Description is updated to indicate that *errno* will not be changed if the function is successful.

umask( )

Set and get the file mode creation mask.

```
#include <sys/types.h>
#include <sys/stat.h>

mode_t umask(mode_t cmask);
```

Issue 5: No functional changes in this issue.

uname( )

Get the name of the current system.

```
#include <sys/utsname.h>

int uname(struct utsname *name);
```

Issue 5: No functional changes in this issue.

ungetc( )

Push a byte back into the input stream.

```
#include <stdio.h>

int ungetc(int c, FILE *stream);
```

Issue 5: No functional changes in this issue.

ungetwc( )

Push a wide-character code back into the input stream.

```
#include <stdio.h>
#include <wchar.h>

wint_t ungetwc(wint_t wc, FILE *stream);
```

Issue 5: The Optional Header (OH) marking is removed from **<stdio.h>**.

unlink( )

Remove a directory entry.

```
#include <unistd.h>

int unlink(const char *path);
```

Issue 5: The [EBUSY] error is added to the ‘‘may fail’’ part of the Errors section.

unlockpt( )

Unlock a pseudo-terminal master/slave pair.

```
#include <stdlib.h>

int unlockpt(int fildes);
```

Issue 5: No functional changes in this issue.

usleep( )

Suspend execution for an interval.

```
#include <unistd.h>

int usleep(useconds_t useconds);
```

Issue 5: The Description is changed to indicate that timers are now thread-based rather than process-based.

utime( )

Set the file access and modification times.

```
#include <sys/types.h>
#include <utime.h>

int utime(const char *path, const struct utimbuf *times);
```

Issue 5: No functional changes in this issue.

utimes( )

Set the file access and modification times.

```
#include <sys/time.h>

int utimes(const char *path, const struct timeval times[2]);
```

Issue 5: No functional changes in this issue.

valloc( )

Page-aligned memory allocator. (**LEGACY**)

```
#include <stdlib.h>

void *valloc(size_t size);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations.

A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *valloc*( ) when access to the function is serialized.

va_arg( ), va_end( ), va_start( )

Handle a variable argument list.

```
#include <stdarg.h>

type va_arg(va_list ap, type);
void va_end(va_list ap);
void va_start(va_list ap, argN);
```

Issue 5: No functional changes in this issue.

vfork( )

Create a new process; share virtual memory.

```
#include <unistd.h>

pid_t vfork(void);
```

Issue 5: No functional changes in this issue.

vfprintf( ), vprintf( ), vsnprintf( ), vsprintf( )

Format the output of a stdarg argument list.

```
#include <stdarg.h>
#include <stdio.h>

int vfprintf(FILE *stream, const char *format, va_list ap);
int vprintf(const char *format, va_list ap);
int vsnprintf(char *s, size_t n, const char *format, va_list ap);
int vsprintf(char *s, const char *format, va_list ap);
```

Issue 5: The *vsnprintf*( ) function is added.

The *vsnprintf*( ) function is the same as *snprintf*( ) except that instead of being called with a variable number of arguments, it is called with an argument list as defined by **<stdarg.h>**.

vfwprintf( ), vwprintf( ), vswprintf( )

Wide-character formatted output of a stdarg argument list.

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vwprintf(const wchar_t *format, va_list arg);
int vfwprintf(FILE *stream, const wchar_t *format, va_list arg);
int vswprintf(wchar_t *s, size_t n, const wchar_t *format,
    va_list arg);
```

Issue 5: These are new functions included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E). These are the wide-character equivalents of *vprintf*( ), *vfprintf*( ), and *vsprintf*( ).

vsprintf( ), vsnprintf( )

Print formatted output.

```
#include <stdarg.h>
#include <stdio.h>

int vsprintf(char *s, const char *format, va_list ap);
int vsnprintf(char *s, size_t n, const char *format, va_list ap);
```

Issue 5: The *vsnprintf*( ) is new. See *vfprintf*( ).

wait( ), waitpid( )

Wait for a child process to stop or terminate.

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *stat_loc);
pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

Issue 5: The Description is updated for alignment with the POSIX Threads Extension. Specifically, the semantics are now described for threads.

wait3( )

Wait for a child process to change state. (**LEGACY**)

```
#include <sys/wait.h>

pid_t wait3 (int *stat_loc, int options, struct rusage *resource_usage);
```

Issue 5: This function is marked LEGACY and may not be available on all implementations. New applications should use *waitpid*( ).

A note indicating that this interface need not be reentrant is added to the Description. A portable multi-threaded application may only safely call *wait3*( ) when access to the function is serialized.

waitid( )

Wait for a child process to change state.

```
#include <sys/wait.h>

int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
```

Issue 5: The Description is updated for alignment with the POSIX Threads Extension.

waitpid( )

Wait for a child process to stop or terminate.

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

Issue 5: No functional changes in this issue.

wcrtomb( )

Convert a wide-character code to a character (restartable).

```
#include <stdio.h>

size_t wcrtomb(char *s, wchar_t wc, mbstate_t *ps);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

wcscat( )

Concatenate two wide-character strings.

```
#include <wchar.h>

wchar_t *wcscat(wchar_t *ws1, const wchar_t *ws2);
```

Issue 5: No functional changes in this issue.

wcschr( )

Wide-character string scanning operation.

```
#include <wchar.h>

wchar_t *wcschr(const wchar_t *ws, wchar_t wc);
```

Issue 5: No functional changes in this issue.

wcscmp( )

Compare two wide-character strings.

```
#include <wchar.h>

int wcscmp(const wchar_t *ws1, const wchar_t *ws2);
```

Issue 5: No functional changes in this issue.

wcscoll( )

Wide-character string comparison using collating information.

```
#include <wchar.h>

int wcscoll(const wchar_t *ws1, const wchar_t *ws2);
```

Issue 5: The Description is updated to indicate that *errno* will not be changed if the function is successful.

wcscpy( )

Copy a wide-character string.

```
#include <wchar.h>

wchar_t *wcscpy(wchar_t *ws1, const wchar_t *ws2);
```

Issue 5: No functional changes in this issue.

wcscspn( )

Get the length of a complementary wide substring.

```
#include <wchar.h>

size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);
```

Issue 5: The Return Value section is updated to indicate that *wcscspn*( ) returns the length of *ws1*, rather than *ws1* itself.

wcsftime( )

Convert date and time to a wide-character string.

```
#include <wchar.h>

size_t wcsftime(wchar_t *wcs, size_t maxsize, const wchar_t *format,
     const struct tm *timptr);
```

Issue 5: Aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of the format argument is changed from **const char*** to **const wchar_t***.

wcslen( )

Get a wide-character string length.

```
#include <wchar.h>

size_t wcslen(const wchar_t *ws);
```

Issue 5: No functional changes in this issue.

wcsncat( )

Concatenate part of two wide-character strings.

```
#include <wchar.h>

wchar_t *wcsncat(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

Issue 5: No functional changes in this issue.

wcsncmp( )

Compare part of two wide-character strings.

```
#include <wchar.h>

int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
```

Issue 5: No functional changes in this issue.

wcsncpy( )

Copy part of a wide-character string.

```
#include <wchar.h>

wchar_t *wcsncpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

Issue 5: No functional changes in this issue.

wcspbrk( )

Scan a wide-character string for a wide-character code.

```
#include <wchar.h>

wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);
```

Issue 5: No functional changes in this issue.

wcsrchr( )

Wide-character string scanning operation.

```
#include <wchar.h>

wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);
```

Issue 5: No functional changes in this issue.

wcsrtombs( )

Convert a wide-character string to a character string (restartable).

```
#include <wchar.h>

size_t wcsrtombs(char *dst, const wchar_t **src, size_t len,
    mbstate_t *ps);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The *wcsrtombs*( ) function converts a sequence of wide characters from the array indirectly pointed to by *src* into a sequence of corresponding characters, beginning in the conversion state described by the object pointed to by *ps*.

wcsspn( )

Get the length of a wide substring.

```
#include <wchar.h>

size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);
```

Issue 5: The Return Value section is updated to indicate that *wcsspn*( ) returns the length of *ws1*, rather that *ws1* itself.

wcsstr( )

Find a wide-character substring.

```
#include <wchar.h>

wchar_t *wcsstr(const wchar_t *ws1, const wchar_t *ws2);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The *wcsstr*( ) function locates the first occurrence in the wide-character string pointed to by *ws1* of the sequence of wide characters (excluding the terminating null wide character) in the wide-character string pointed to by *ws2*.

In an earlier draft of the ISO/IEC 9899:1990/Amendment 1:1995 (E) the *wcsstr*( ) function was named *wcswcs*( ). Applications using *wcswcs*( ) should be rewritten

to use *wcsstr*( ).

wcstod( )

Convert a wide-character string to a double-precision number.

```
#include <wchar.h>

double wcstod(const wchar_t *nptr, wchar_t **endptr);
```

Issue 5: The Description is updated to indicate that *errno* will not be changed if the function is successful.

wcstok( )

Split a wide-character string into tokens.

```
#include <wchar.h>

wchar_t *wcstok(wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr);
```

Issue 5: Aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, a third argument is added to the definition of this function in the Synopsis.

wcstol( )

Convert a wide-character string to a long integer.

```
#include <wchar.h>

long int wcstol(const wchar_t *nptr, wchar_t **endptr, int base);
```

Issue 5: The Description is updated to indicate that *errno* will not be changed if the function is successful.

wcstombs( )

Convert a wide-character string to a character string.

```
#include <stdlib.h>

size_t wcstombs(char *s, const wchar_t *pwcs, size_t n);
```

Issue 5: No functional changes in this issue.

wcstoul( )

Convert a wide-character string to an **unsigned long**.

```
#include <wchar.h>

unsigned long int wcstoul(const wchar_t *nptr, wchar_t **endptr,
     int base);
```

Issue 5: The Description is updated to indicate that *errno* will not be changed if the function is successful.

wcswcs( )

Find a wide substring.

```
#include <wchar.h>

wchar_t *wcswcs(const wchar_t *ws1, const wchar_t *ws2);
```

Issue 5: Marked as EX.

wcswidth( )

Number of column positions of a wide-character string.

```
#include <wchar.h>

int wcswidth(const wchar_t *pwcs, size_t n);
```

Issue 5: No functional changes in this issue.

wcsxfrm( )

Wide-character string transformation.

```
#include <wchar.h>

size_t wcsxfrm(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

Issue 5: The Description is updated to indicate that *errno* will not be changed if the function is successful.

wctob( )

Wide-character to single-byte conversion.

```
#include <stdio.h>
#include <wchar.h>

int wctob(wint_t c);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The *wctob*( ) function determines whether *c* corresponds to a member of the extended character set whose character representation is a single byte when in the initial shift state.

wctomb( )

Convert a wide-character code to a character.

```
#include <stdlib.h>

int wctomb(char *s, wchar_t wchar);
```

Issue 5: No functional changes in this issue.

wctrans( )

Define character mapping.

```
#include <wctype.h>

wctrans_t wctrans(const char *charclass);
```

Issue 5: This is a new function derived from the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The *wctrans*( ) function is defined for valid character mapping names identified in the current locale.

wctype( )

Define character class.

```
#include <wctype.h>

wctype_t wctype(const char *property);
```

Issue 5: The following change has been made in this issue for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):

- The Synopsis has been changed to indicate that this function and associated data types are now made visible by inclusion of the header **<wctype.h>**, rather than **<wchar.h>**.

wcwidth( )

Number of column positions of a wide-character code.

```
#include <wchar.h>

int wcwidth(wchar_t wc);
```

Issue 5: No functional changes in this issue.

wmemchr( )

Find a wide character in memory.

```
#include <wchar.h>

wchar_t *wmemchr(const wchar_t *ws, wchar_t wc, size_t n);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The *wmemchr*( ) function locates the first occurrence of *wc* in the initial *n* wide characters of the object pointed to be *ws*.

wmemcmp( )

Compare wide characters in memory.

```
#include <wchar.h>

int wmemcmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The *wmemcmp*( ) function compares the first *n* wide characters of the object pointed to by *ws1* to the first *n* wide characters of the object pointed to by *ws2*.

wmemcpy( )

Copy wide characters in memory.

```
#include <wchar.h>

wchar_t *wmemcpy(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The *wmemcpy*( ) function copies *n* wide characters from the object pointed to by *ws2* to the object pointed to by *ws1*.

wmemmove( )

Copy wide characters in memory with overlapping areas.

```
#include <wchar.h>

wchar_t *wmemmove(wchar_t *ws1, const wchar_t *ws2, size_t n);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The *wmemmove*( ) function copies *n* wide characters from the object pointed to by *ws2* to the object pointed to by *ws1*.

wmemset( )

Set wide characters in memory.

```
#include <wchar.h>

wchar_t *wmemset(wchar_t *ws, wchar_t wc, size_t n);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

The *wmemset*( ) function copies the value of *wc* into each of the first *n* wide characters of the object pointed to by *ws*.

wordexp( ), wordfree( )

Perform word expansions.

```
#include <wordexp.h>

int wordexp(const char *words, wordexp_t *pwordexp, int flags);
void wordfree(wordexp_t *pwordexp);
```

Issue 5: No functional changes in this issue.

wprintf( )

Print formatted wide-character output.

```
#include <stdio.h>
#include <wchar.h>

int wprintf(const wchar_t *format, ...);
```

Issue 5: Included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

This is the wide-character equivalent of *printf*( ).

write( ), writev( ), pwrite( )

Write on a file.

```
#include <unistd.h>

ssize_t write(int fildes, const void *buf, size_t nbyte);
ssize_t pwrite(int fildes, const void *buf, size_t nbyte,
    off_t offset);

#include <sys/uio.h>

ssize_t writev(int fildes, const struct iovec *iov, int iovcnt);
```

Issue 5: The Description is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Large File Support extensions are added; see Chapter 17 on page 163.

The *pwrite*( ) function is added.

wscanf( )

Convert formatted wide-character input.

```
#include <stdio.h>
#include <wchar.h>

int wscanf(const wchar_t *format, ...);
```

Issue 5: This is a new function included for alignment with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

This is the wide-character equivalent of *scanf*( ).

y0( ), y1( ), yn( )

Bessel functions of the second kind.

```
#include <math.h>

double y0(double x);
double y1 (double x);
double yn (int n, double x);
```

Issue 5: No functional changes in this issue.